Tabla de Contenido

Tabla de Figuras

Capítulo 1. Introducción

Sistemas de Gestión de Bases de Datos y Lenguajes de Bases de Datos

SGBD: Sistema de Gestión de Bases de Datos, son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos y el usuario. Los Sistemas Gestores de Bases de Datos proporcionan un interfaz entre aplicaciones y sistema operativo, consiguiendo, entre otras cosas, que el acceso a los datos se realice de una forma más eficiente, más fácil de implementar y, sobre todo, más segura.

1.1. ¿Qué se espera de un SGBD?

De un Sistema de Gestión de Bases de Datos esperamos:

- Permita a los usuarios crear otras bases de datos y especificar su esquema por medio de algún lenguaje de definición
- Ofrezca a los usuarios la capacidad de consultar los datos y modificarlos, usando para ello un lenguaje de consulta y manipulación.
- Brinde soporte al almacenamiento de cantidades voluminosas de datos durante un largo período, protegiéndolo contra accidentes o utilización no autorizada.
- Controle el acceso concurrente

1.2. Arquitectura mínima de un SGBD

Elementos mínimos de un SGBD son:

- **Procesador / Administrador de Consultas**, traducción y chequeo de las consultas de los usuarios.
- **Administrador de transacciones**, debe facilitar un mantenimiento de las propiedades ACID:
    - **Atomicity**: Atomicidad (o todas las operaciones se realizan o ninguna)
    - **Consystency**, Consistencia: El estado de la BD (invariante) es consistente antes y después de cada transacción.
    - **Isolation**: Aislamiento. Las operaciones concurrentes lucen secuenciales
    - **Durability**: Los cambios comprometidos perduran en el tiempo
- **Administrador de Almacenamiento**: Se encarga de administrar los archivos físicos de la BD y el buffer (memoria intermedia).
- **Repositorio** de metadatos y datos.

1.3. Lenguajes de Bases de Datos

Además, un SGBD debe tener un lenguaje con el que poder trabajar con él, el más conocido es el **SQL (Structured Query Language)** en él se integra un DDL y un DML:

- Lenguajes de definición de datos (DDL): Creación de esquemas, modificación de los mismos, etc. Sus resultados se almacenan en el diccionario de datos.
- Lenguaje de manipulación de datos (DML): Creación, Modificación, Eliminación y Obtención de Datos.

1.4. ¿Qué exigiremos a un SGBD?

Para optar por un buen SGBD debe estudiar su:

- **Escalabilidad**: Capacidad de mejorar con el incremento de los recursos invertidos.
- **Portabilidad**: Exportación e importación de datos de una plataforma a otra.
- **Rendimiento**: Recuperación, actualización, concurrencia, etc. de una manera eficiente.
- **Universalidad**: Múltiples tipos de datos (multimedia).
- **Disponibilidad**: 7x24.

1.5. Estructura de un SGBD

El siguiente gráfico resume los elementos que componen un SGBD:

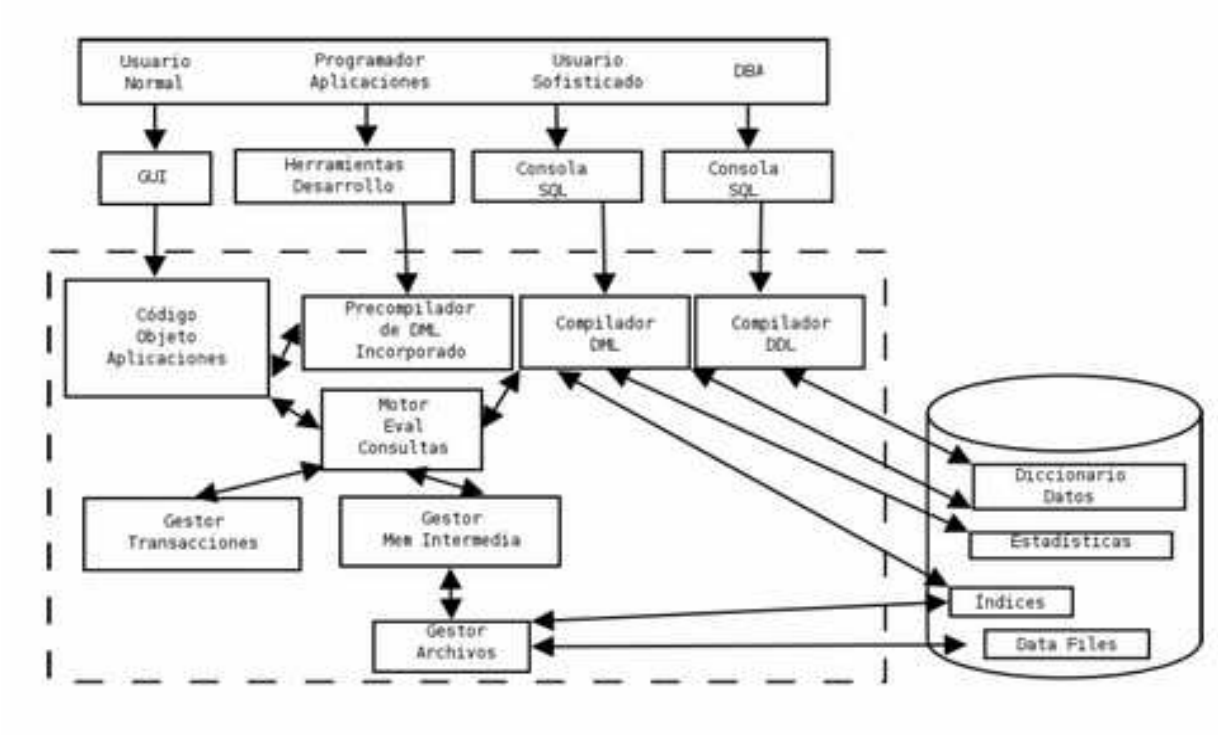

Figura 1-1: Estructura de un SGBD

Capítulo 2. Características principales

PostgreSQL is un potente sistema de base de datos relacional libre (open source, su código fuente está disponible) liberado bajo licencia BSD. Tiene mas de 15 años de activo desarrollo y arquitectura probada que se ha ganado una muy buena reputación por su confiabilidad e integridad de datos. Funciona en todos los sistemas operativos importantes , incluyendo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows.

El desarrollo de PostgreSQL es realizado por un equipo de desarrolladores (voluntarios en su mayoría) dispersos alrededor del mundo y comunicados vía Internet. Este es un proyecto de la comunidad y no es controlado por ninguna compañía. Para integrarse al proyecto vea el FAQ de los desarrolladores:

[http://www.postgresql.org/files/documentation/faqs/FAQ_DEV.html].

PostgreSQL es un servidor de base de datos relacional libre, liberado bajo la licencia BSD. Es una alternativa a otros sistemas de bases de datos de código abierto (como MySQL, Firebird y MaxDB), así como sistemas propietarios como Oracle o DB2.

2.1. Principales características

- soporta casi toda la **sintaxis SQL** tiene soporte total para foreign keys, joins, views, triggers, y stored procedures (en multiples lenguajes).
- **Integridad transaccional**, obedece completamente a la especificación ACID.
- Acceso concurrente multiversión, **MVCC Control de Concurrencia Multi-Versión** (Multi-Version Concurrency Control), no se bloquean las tablas, ni siquiera las filas, cuando un proceso escribe. Es la tecnología que PostgreSQL usa para evitar bloqueos innecesarios. Mediante el uso de MVCC, PostgreSQL evita el problema de que procesos lectores estén esperando a que se termine de escribir. En su lugar, PostgreSQL mantiene una ruta a todas las transacciones realizadas por los usuarios de la base de datos. PostgreSQL es capaz entonces de manejar los registros sin necesidad de que los usuarios tengan que esperar a que los registros estén disponibles.
- **Cliente/Servidor**: PostgreSQL usa una arquitectura proceso-por-usuario cliente/servidor. Esta es similar al método del Apache 1.3.x para manejar procesos. Hay un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que intente conectar a PostgreSQL.
- **Write Ahead Logging (WAL)**: La característica de PostgreSQL conocida como Write Ahead Logging incrementa la dependencia de la base de datos al registro de cambios antes de que estos sean escritos en la base de datos. Esto garantiza que en el hipotético caso de que la base de datos se caiga, existirá un registro de las transacciones a partir del cual podremos restaurar la base de datos. Esto puede ser enormemente beneficioso en el caso de caída, ya que cualesquiera cambios que no fueron escritos en la base de datos pueden ser recuperados usando el dato que fue previemante registrado. Una vez el sistema ha quedado restaurado, un usuario puede continuar trabajando desde el punto en que lo dejó cuando cayó la base de datos.

- **Lenguajes Procedurales**: PostgreSQL tiene soporte para lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL. Este lenguaje es comparable al lenguaje procedural de Oracle, PL/SQL. Otra ventaja de PostgreSQL es su habilidad para usar Perl, Python, o TCL como lenguaje procedural embebido. además de en C, C++ y, Java.
- **Interfaces con lenguajes de programación**. La flexibilidad del API de PostgreSQL ha permitido a los vendedores proporcionar soporte al desarrollo fácilmente para el RDBMS PostgreSQL. Estas interfaces incluyen Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++, Pike, etc.
- **Herencia** de tablas.
- Incluye la mayoría de los **tipos de datos SQL92 y SQL99** (INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, y TIMESTAMP), soporta almacenamiento de objetos grandes binarios, además de tipos de datos y operaciones geométricas.
- **Puntos de recuperación** a un momento dado, tablespaces, replicación asincrónica, transacciones jerarquizadas (savepoints), backups en línea.
- Un sofisticado **analizador/optimizador de consultas.**
- Soporta **juegos de caracteres internacionales**, codificación de caracteres multibyte.
-

2.2. Límites

Es importante tener en cuanta las limitaciones que tiene PostgreSQL:

- Máximo tamaño de base de datos ilimitado.
- Máximo tamaño de tabla 32 TB.
- Máximo tamaño de tupla 1.6 TB.
- Máximo tamaño de campo 1 GB.
- Máximo tuplas por tabla ilimitado.
- Máximo columnas por tabla 250 - 1600 dependiendo de los tipos de columnas.
- Máximo de índices por tabla ilimitado.

Capítulo 3. Breve historia

Nota: La reseña histórica que se cita a continuación se ha obtenido de Wikipedia [http://es.wikipedia.org/wiki/PostgreSQL] y de la documentación aportada por el equipo de desarrollo de PostgreSQL [http://es.tldp.org/Postgresql-es/web/navegable/tutorial/x56.html#AEN64].

3.1. De Ingres a Postgres

PostgreSQL es el último resultado de una larga evolución comenzada con el proyecto Ingres en la Universidad de Berkeley. El líder del proyecto, *Michael Stonebraker* abandonó **Berkeley** para comercializar Ingres en 1982, pero finalmente regresó a la universidad y, en 1985, Stonebraker comenzó un proyecto *post-Ingres* para resolver los problemas con el modelo de base de datos relacional que habían sido aclarados a comienzos de los años 80. El principal de estos problemas era la incapacidad del modelo relacional de comprender "tipos", es decir, combinaciones de datos simples que conforman una única unidad. Actualmente estos son llamados objetos.

El proyecto resultante, llamado **Postgres**, era orientado a introducir la menor cantidad posible de funcionalidades para completar el soporte de tipos. Estas funcionalidades incluían la habilidad de definir tipos, pero también la habilidad de describir relaciones. En Postgres la base de datos "comprendía" las relaciones y podía obtener información de tablas relacionadas utilizando reglas.

El Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos conocido como PostgreSQL (y brevemente llamado Postgres95) está derivado del paquete Postgres escrito en Berkeley.

3.2. El proyecto Postgres de Berkeley

La implementación del DBMS Postgres comenzó en 1986. En distintos documentos se presentaron los conceptos iniciales para el sistema, la definición del modelo de datos inicial y el diseño del sistema de reglas, así como se detallaron la la lógica y arquitectura del gestor de almacenamiento.

Postgres ha pasado por varias revisiones importantes desde entonces. El primer sistema de pruebas fue operacional en 1987. Se lanzó la Versión 1 en Junio de 1989. En respuesta a una crítica del primer sistema de reglas, éste fue rediseñado en la Versión 2, que salió en Junio de 1990. La Versión 3 apareció en 1991 y añadió una implementación para múltiples gestores de almacenamiento, un ejecutor de consultas mejorado y un sistema de reescritura de reglas nuevo. En su mayor parte, las siguientes versiones hasta el lanzamiento de Postgres95 (ver más abajo) se centraron en mejorar la portabilidad y la fiabilidad.

El tamaño de la comunidad de usuarios externos casi se duplicó durante 1993. Pronto se hizo obvio que el mantenimiento del código y las tareas de soporte estaban ocupando tiempo que debía dedicarse a la investigación. En un esfuerzo por reducir esta carga, el proyecto terminó oficialmente con la Versión 4.2.

3.3. Postgres95

En 1994, Andrew Yu y Jolly Chen añadieron un intérprete de lenguage SQL a Postgres. **Postgres95** fue publicado a continuación en la Web para que encontrara su propio hueco en el mundo como un descendiente de dominio público y código abierto del código original Postgres de Berkeley.

El código de Postgres95 fue adaptado a ANSI C y su tamaño reducido en un 25%. Además de corrección de errores, éstas fueron las principales mejoras:

- El lenguage de consultas Postquel fue reemplazado con SQL (implementado en el servidor).. La interfaz libpq permaneció disponible para programas escritos en C.
- Además del programa de monitorización, se incluyó un nuevo programa (psql) para realizar consultas SQL interactivas usando la librería GNU readline.
- Una nueva librería de interfaz, libpgtcl, soportaba clientes basados en Tcl. Un shell de ejemplo, pgtclsh, aportaba nuevas órdenes Tcl para interactuar con el motor Postgres95 desde programas tcl.
- Se revisó la interfaz con objetos grandes. Los objetos grandes de Inversion fueron el único mecanismo para almacenar objetos grandes (el sistema de archivos de Inversion fue eliminado).
- Se eliminó también el sistema de reglas a nivel de instancia, si bien las reglas siguieron disponibles como reglas de reescritura.
- Se distribuyó con el código fuente un breve tutorial introduciendo las características comunes de SQL y de Postgres95.
- Se utilizó GNU make (en vez de BSD make) para la compilación. Postgres95 también podía ser compilado con un gcc sin parches (al haberse corregido el problema de alineación de variables de longitud doble).
-

3.4. PostgreSQL

En 1996, se hizo evidente que el nombre "Postgres95" no resistiría el paso del tiempo. Elegieron un nuevo nombre, **PostgreSQL**, para reflejar la relación entre el Postgres original y las versiones más recientes con capacidades SQL. Al mismo tiempo, se hicieron que los números de versión partieran de la 6.0, volviendo a la secuencia seguida originalmente por el proyecto Postgres.

Durante el desarrollo de Postgres95 se hizo hincapié en identificar y entender los problemas en el código del motor de datos. Con PostgreSQL, el énfasis ha pasado a aumentar características y capacidades, aunque el trabajo continúa en todas las áreas.

Las principales mejoras en PostgreSQL incluyen:

- Los bloqueos de tabla han sido sustituidos por el control de concurrencia multi-versión, el cual permite a los accesos de sólo lectura continuar leyendo datos consistentes durante la actualización de registros, y permite copias de seguridad en caliente desde pg_dump mientras la base de datos permanece disponible para consultas.
- Se han implementado importantes características del motor de datos, incluyendo subconsultas, valores por defecto, restricciones a valores en los campos (constraints) y disparadores (triggers).

- Se han añadido funcionalidades en línea con el estándar SQL92, incluyendo claves primarias, identificadores entrecomillados, forzado de tipos cadena literales, conversión de tipos y entrada de enteros binarios y hexadecimales.
- Los tipos internos han sido mejorados, incluyendo nuevos tipos de fecha/hora de rango amplio y soporte para tipos geométricos adicionales.
- La velocidad del código del motor de datos ha sido incrementada aproximadamente en un 20-40%, y su tiempo de arranque ha bajado el 80% desde que la versión 6.0 fue lanzada.

Capítulo 4. Herramientas y programas

Herramientas y programas para trabajar con PostgreSQL.

Se pretende en este capítulo hacer una breve mención de programas y herramientas que se pueden usar para trabajar con PostgreSQL, algunas de ellas vienen con la propia distribución, otras han sido creadas por terceros, así como unas son para usar en clientes que se conectan a un servidor, y otras son para usarlas en un servidor. Además, se comentan algunos proyectos interesantes, así como se entra en más detalle en los dos programas más conocidos para conectarse a PostgreSQL.

4.1. Software proporcionado por la propia distribución de PostgreSQL

4.1.1. Aplicaciones cliente

Casi todas las aplicaciones se corresponden con una órden de SQL ejecutadas dentro de la base de datos, para saber más detalles, consultar el manual de cada programa tal como se hace en Linux / Unix:

```
$ <comando> man
```

- **clusterdb**: equivalente al comando **CLUSTER** de SQL, reorganiza cluster de tablas.
- **createdb**: crea una base de datos.
- **createlang**: define un nuevo lenguaje procedural en una base de datos.
- **createuser**: creación de usuarios.
- **dropdb**: borrar una base de datos.
- **droplang**: borrar un lenguaje procedural.
- **dropuser**: borrar un usuario.
- **ecpg:** SQL C preprocesador embebido.
- **pg_config**: recupera información sobre la versión instalada de PostgreSQL.
- **pg_dump**: extrae una base de datos en un fichero script o en otros formatos.
- **pg_dumpall**: extrae un cluster de base de datos en un fichero script.
- **pg_restore**: restaura una base de datos desde un fichero creado con pg_dump.
- **psql**: terminal interactivo de PostgreSQL. Es la herramienta canónica para la ejecución de sentencias SQL a través del shell del SO Es una herramienta de tipo frontend que permite describir sentencias SQL, ejecutarlas y visualizar sus resultados El método de ingreso puede ser mediante la inserción directa del código en la consola, o la ejecución de sentencias dentro de un archivo de texto Provee de diversos meta-comandos para la ejecución de las sentencias, así como diversas opciones tipo shell propias de la herramienta

- **reindexdb**: reindexa una base de datos.
- **vacuumdb**: reorganiza y analiza una base de datos.

4.1.2 Aplicaciones servidor

- **initdb**: crea un cluster de base de datos.
- **ipcclean**: libera la memoria compartida y los semáforos de un servidor PostgreSQL.
- **pg_controldata**: muestra información de control de un cluster de base de datos.
- **pg_ctl**: inicia, para o reinicia un servidor PostgreSQL.
- **pg_resetxlog**: reinicia el “write-ahead log” y otras informaciones de control de un cluster de base de datos.
- **postgres**: corre un servidor PostgreSQL en modo "single-user".
- **postmaster**: servidor de base de datos PostgreSQL multiusuario.

4.2. Software libre de terceros

Se citan en este apartado herramientas de Software Libre, en el mundo hay muchas más herramientas comerciales.

4.2.1. Programas

- **pgAdmin III**:
  - herramienta gráfica, permite ver la estructura de las bases de datos, realizar operaciones SQL, ver datos, operaciones de administración, para más detalles ver ???
  - diseñada para ejecutarse en muchos sistemas operativos (Windows, Linux, MacOS).
- **phppgAdmin III**: PHPPgAdmin es un poderosa herramienta de administración basada en un interfaz Web para bases de datos PostgreSQL.
  - Equivalente a la anterior herramienta, pero está realizada en una interfaz web con PHP.
  - Tiene la ventaja de que no requiere la instalación en los clientes, así como se puede centralizar la conexión a las bases de datos, impidiendo el acceso desde estaciones de trabajo y, a la vez, facilitando el trabajo a los potenciales usuarios porque no tienen que dedicar tiempo a configurar conexiones.
  - Dispone de soporte para procedimientos almacenados, triggers y vistas.
  - Para más información, ver sede web [http://phppgadmin.sourceforge.net/].

- **PgAccess**:
  - Es una interfaz gráfica para el gestor de bases de datos PostgreSQL escrito por Constantin Teodorescu en el lenguaje Tcl/Tk.
  - Permite al usuario interactuar con PostgreSQL de una manera similar a muchas aplicaciones de bases de datos para PC, con menús de opciones y diversas herramientas gráficas. Esto significa que el usuario puede evitar la linea de comandos para la mayoría de las tareas.
  - Tiene opciones para creación de formularios e informes.
  - Ver sede web [http://www.pgaccess.org].
- **pgExplorer**:
  - Herramienta de desarrollo para Postgres con una amplia interfaz gráfica.
  - Se incluye un vista en árbol de las bases de datos y sus respectivos objetos.
  - Permite ingeniería inversa.
  - Tiene asistentes para generar sentencias SQL.
  - Incluye diseñador gráfico de consultas.
  - Ver sede web [http://www.pgexplorer.org].
- Herramientas de acceso mediante **OBDC** y **JDBC**: se permite la conexión mediante estos drivers, que existen para casi todas las plataformas, con lo que es fácil conectar desde cualquier producto que soporte ODBC/JDBC y tengamos los drivers adecuados.

4.2.2. Proyectos

Se citan aquí algunos proyectos donde se están desarrollando utilidades, programas, mejoras muy interesantes:

- pgfoundry [http://pgfoundry.org/]: sitio web donde está el repositorio de proyectos de PostgreSQL que no se incluyen dentro del núcleo.
- S-lony [http://gborg.postgresql.org/project/slony1/projdisplay.php]: herramienta para replicación de bases de datos PostgreSQL.
- PGCluster [http://pgcluster.projects.postgresql.org/feature.html]: proyecto para formar cluster de bases de datos con PostgreSQL.
- GiST: Generalized Search Tree, "Árbol de Búsqueda Generalizado". Mecanismo extensible de indexamiento, mejorado de manera que soporta concurrencia de alta velocidad, recuperabilidad y rendimiento de actualizaciones, que antes estaba disponible sólo para los índices B-Tree. GiST es el mecanismo central para el soporte de indexamiento total de texto (TSearch2), geoespacial (GIS) y de estructuras de árbol. Con esta mejora, los tipos de datos complejos tendrán buen rendimiento aún en grandes aplicaciones de alta disponibilidad.

- PostGIS [http://www.postgis.org]: módulo que añade soporte de objetos geográficos a la base de datos PostgreSQL para su uso para Sistemas de Información Geográfica. El proyecto gvSIG [http://www.gvsig.gva.es/] de la Generalitat Valenciana se realiza con PostGIS + PostgreSQL.

Capítulo 5. Estructura de PostgreSQL I

Conceptos: “Servidor”, “Proceso Servidor”, “Proceso Cliente”, “Aplicaciones Cliente”, “Bases de datos”, “Cluster de bases de datos”, “Instancia”, “Espacio en disco”.

5.1. Arquitectura

PostgreSQL funciona con una arquitectura Cliente/Servidor, un proceso servidor (**postmaster**) y una serie de aplicaciones cliente que realizan solicitudes de acciones contra la base de datos a su proceso servidor. Por cada una de estas aplicaciones cliente, el proceso postmaster crea un proceso **postgres.**

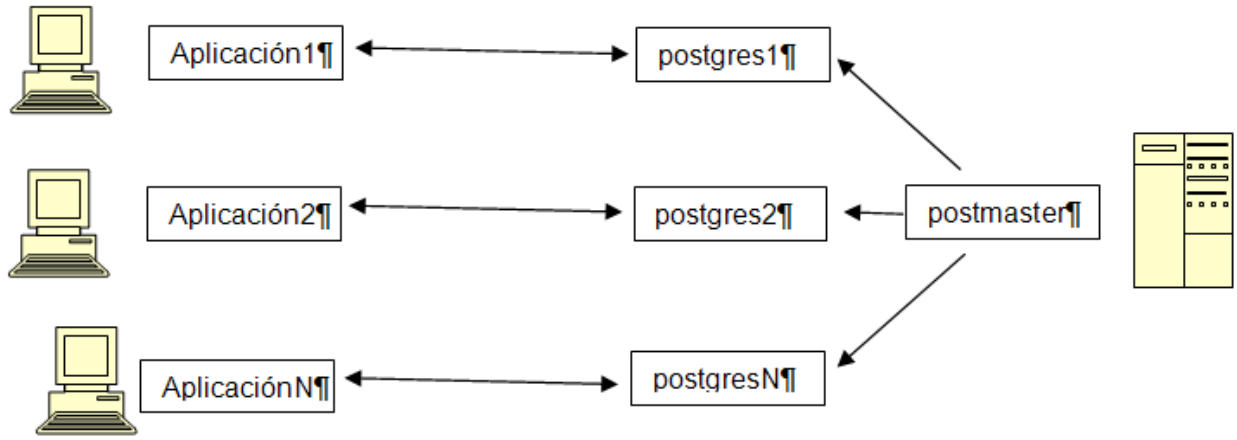

Figura 5-1: Arquitectura PostgreSQL

Tenemos aquí que se ejecutan una serie de aplicaciones cliente (*FrontEnd*) y una serie de procesos en el servidor (*BackEnd*), de modo que, por ejemplo, la interacción entre un programa que se ejecuta en un cliente, por ejemplo, pgadmin3, una aplicación PHP, etc y el servidor de base de datos sería:

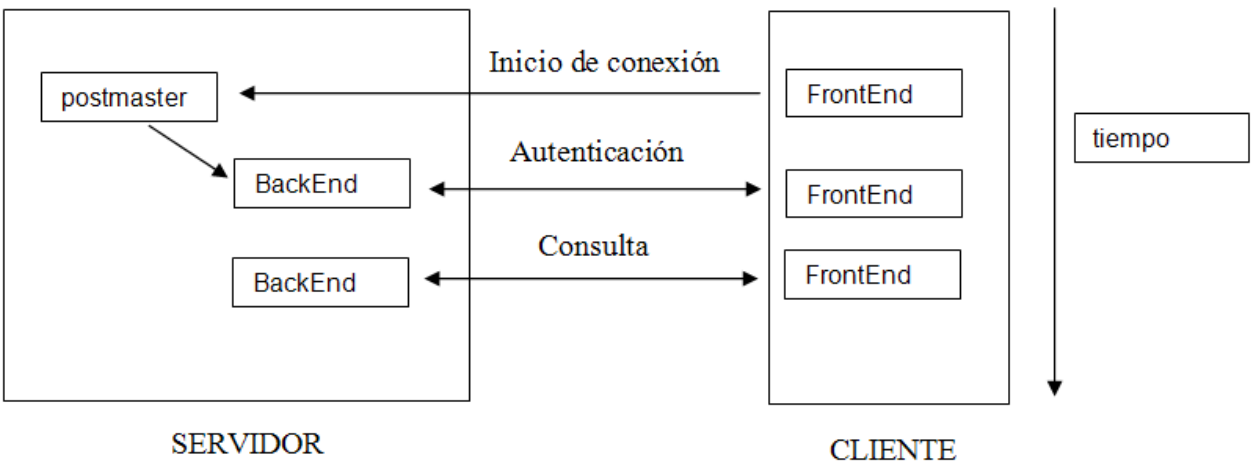

Figura 5-2: Esquema de conexión entre procesos FrontEnd/BackEnd

El **proceso Postmaster**:

- es el proceso inicial
- gestiona los accesos multiusuario y multiconexión
- levanta la memoria compartida
- está al tanto de solicitudes de nuevas conexiones
- lanza procesos de atención de demanda, realizando las operaciones sobre la base de datos a solicitud de los clientes
- realiza el enlazado con los archivos de datos, gestionando estos ficheros, donde los ficheros pueden pertenecer a varias bases de datos

La comunicación entre BackEnd/FrontEnd se realiza mediante **TCP/IP**, normalmente por el **puerto 5432**, creando un **socket (/tmp/.s.PGSQL.5432).**

La **memoria compartida**:

- gestiona los recursos entre procesos backend
- gestiona la caché del disco
- maneja otras estructuras internas

De esta manera podemos definir el concepto de **CLUSTER DE BASE DE DATOS**, como una instancia de PostgreSQL, donde se lanza un proceso postmaster, que puede gestionar varias bases de datos. En un servidor, pueden haber varios cluster, cada uno tendrá su postmaster, y cada postmaster escuchará por un puerto diferente.

Nota: no confundir este concepto de "cluster" con los típicos que se manejan en bases de datos de "base de datos en cluster" o "tablas en cluster".

5.2 Almacenamiento físico

En un servidor se crean uno o varios clusters de bases de datos. La estructura física de del cluster se crea con el programa **initdb**, con este programa se determina la ubicación física y el juego de caracteres.

En las instalaciones automáticas a partir de paquetes (rpm, apt-get, etc) o en Windows, se crea un cluster automáticamente con tres bases de datos, "template0", "template1" y "postgres". El cluster se crea en un directorio **"data"** dentro del directorio donde se ha instalado postgres. Normalmente, se define una variable de entorno, *PGDATA* que apunte al directorio donde se crea el cluster.

La estructura inicial cuando se crea un cluster es:

```
postgres@debian:~/data$ pwd
/home/postgres/data
postgres@debian:~/data$ ls -lahU
total 35K
drwx------  10 postgres postgres 1,0K 2006-06-14 01:28 .
drwxr-xr-x   6 postgres postgres 1,0K 2006-06-14 01:26 ..
drwx------   2 postgres postgres 1,0K 2006-06-14 01:28 global
drwx------   3 postgres postgres 1,0K 2006-06-14 01:26 pg_xlog
drwx------   2 postgres postgres 1,0K 2006-06-14 01:26 pg_clog
drwx------   2 postgres postgres 1,0K 2006-06-14 01:26 pg_subtrans
drwx------   2 postgres postgres 1,0K 2006-06-14 01:26 pg_twophase
drwx------   4 postgres postgres 1,0K 2006-06-14 01:26 pg_multixact
drwx------   5 postgres postgres 1,0K 2006-06-14 01:26 base
drwx------   2 postgres postgres 1,0K 2006-06-14 01:26 pg_tblspc
-rw-------   1 postgres postgres    4 2006-06-14 01:26 PG_VERSION
-rw-------   1 postgres postgres  14K 2006-06-14 01:26 postgresql.conf
-rw-------   1 postgres postgres 3,4K 2006-06-14 01:26 pg_hba.conf
-rw-------   1 postgres postgres 1,5K 2006-06-14 01:26 pg_ident.conf
-rw-------   1 postgres postgres  476 2006-06-14 01:28 server.log
-rw-------   1 postgres postgres   46 2006-06-14 01:28 postmaster.pid
-rw-------   1 postgres postgres   57 2006-06-14 01:28 postmaster.opts
postgres@debian:~/data$ _
```

Figura 5-3: Directorios y ficheros al crear un cluster

5.2.1. Ficheros:

- **postgresql.conf**: fichero de configuración principal, contiene la asignación a los parámetros que configuran el funcionamiento del servidor.
- **pg_hba.conf**: fichero de configuración de la autenticación de los clientes y usuarios y del acceso a las bases de datos del cluster.
- **pg_ident.conf**: fichero accesorio al anterior, determina como se realiza la autenticación *ident* que contiene la correspondencia entre usuarios del Sistema Operativo y de PostgreSQL.
- **PG_VERSION**: fichero de texto con la versión de software Postgres que crea el cluster .
- Otros ficheros:
  - **postmaster.pid**: se crea cuando el postmaster arranca, contiene el PID del proceso postmaster.
  - **postmaster.opts**. contiene las opciones con las que se ha iniciado el postmaster.
  - **recovery.conf, recovery.done**: si se quiere o se ha hecho una recuperación.

5.2.2. Directorios:

- **base**: las plantillas y las bases de datos. contiene un directorio por cada base de datos, dentro hay un fichero por cada tabla o índice de una base de datos, los números corresponden a los OIDs de las tablas o índices.
  - ***template0 (1):*** contiene las definiciones de las tablas del sistema, vistas, funciones y tipos estándar. Nunca se debe modificar ni intentar conectarse a ella, ya que está por si *template1* se corrompe..

    - ***template1 (N):*** base de datos plantilla para crear nuevas bases de datos, se puede modificar su estructura, añadiendo tablas, índices, funciones, etc. Es la que se usará como plantilla para crear otras bases de datos.
- **global**: tablas e indices del catálogo comunes a todas las bases de datos.
    - catálogo compartido: *pg_shadow* (usuarios), *pg_database, etc.*
    - *pgstat.stat:* fichero usado por el monitor de estadísticas.
    - *pg_control*: fichero con parámetros del cluster, algunos inmutables (establecidos en la creación del cluster) y otros variables (establecidos en la puesta en marcha).
- **pg_log**: ficheros de seguimiento del servidor. Se crea en la versión de Windows, en la de Linux, se debe indicar al arrancar el postmaster en qué fichero se hace el seguimiento.
- **pg_xlog**: ficheros de diario del servidor (WAL).
    - Contiene los diarios de escritura adelantada, para usarlos en las recuperaciones
    - se implementan como un conjunto de segmentos (ficheros) de un tamaño de 16Mb y divididos en páginas de 8Kb.
    - Inicialmente se crea un fichero, y luego el sistema va creando más según las necesidades.
- **pg_clog**: ficheros de diario para las transacciones (estado de cada transacción).
    - Contiene los ficheros de confirmación.
    - Un diario de confirmación refleja el estado de cada transacción: confirmada, en progreso o abortada.
- **pg_multixact**: contiene datos sobre el estado multi-transaccional, usado para los bloqueos compartidos de filas.
- **pg_twophase**: ficheros de estado para las transacciones preparadas.
- **pg_subtrans**: para realizar los “savepoints” en medio de transacciones.
- **pg_tblspc**: información sobre los tablespaces. Cuidado porque en linux/unix contiene enlaces a los directorios donde se crean los tablespaces y hay que controlar, en caso de cambios de ficheros, que estén correctos.

5.2.3. Creación del cluster de bases de datos: *initdb*

Después de instalar PostgreSQL, lo primero que hay que realizar en la creación de un cluster, teniendo en cuenta que en un servidor, se pueden crear todos los cluster que se quieran.

```
$ initdb [opciones]
```

Las opciones principales son:

- -D directorio | --d=directorio: directorio donde se crea, si no se indica, usa lo que establezca la variable de entorno *PGDATA*
- -E juego_caracteres | --e=juego_caracteres: si no se indica, coge el del SO
- -U usuario | --u=usuario: superusuario del cluster
- -W clave | --w=clave: establece la palabra de paso del superusuario del cluster

- --locale=ubicación: ubicación geográfica, se puede concretar más:
    - --lc-collate = ubicación: ordenamiento
    - --lc_ctype = ubicación: clasificación de caracteres
    - --lc_messages = ubicación: mensajes
    - --lc-monetary = locale
    - --lc-numeric = locale
    - --lc-time = locale

Sobre los temas de juego de caracteres y localización hay que tener presentes varias cosas:

1. Que el sistema operativo soporte dicha localización o codificación
2. Una vez inicializado el cluster, las variables lc_collate y lc_ctype no se pueden cambiar.
3. Llevar mucho cuidado con los juegos de caracteres y la localización de los equipos que van a trabajar contra la base de datos, sobre todo si se utilizan estaciones Linux para conectarse o manipular ficheros de los servidores.

5.3 Esquema lógico

5.3.1 Conceptos

Concepto de ***"Cluster de bases de datos"*** en PostgreSQL:

- repositorio que engloba un conjunto de bases de datos, que contienen objetos que se pueden guardar en distintos tablespaces y un conjunto de usuarios que se conectan al cluster.
- una base de datos engloba un conjunto de esquemas, los cuales tienen un usuario propietario. En los esquemas es donde se crean los objetos (tablas, índices, procedimientos, vistas, etc.)
- una sesión se abre contra una base de datos

Con lo que tenemos aquí los tres elementos principales a nivel lógico en un cluster:

- ***Bases de datos***: agrupaciones de *esquemas*. Por defecto, siempre hay tres bases de datos creadas, *template0, template1 y postgres.*
- ***Tablespaces***: ubicaciones alternativas a la que por defecto tiene el cluster. Por defecto no se crea ninguno.
- ***Roles***: engloba el concepto de *usuarios (roles de login)* y *grupos de permisos (roles de grupo)*, estos últimos son lo mismo que los roles de Oracle. Hasta la versión 8 de Postgres no se podían anidar roles, ahora si. Por defecto, si al crear el cluster no se ha indicado otro usuario, se crea el usuario *postgres* como superusuario.

Hay que tener en cuenta:

- Todos los usuarios son comunes a las bases de datos del cluster, se pueden conectar con cualquiera de las bases de datos. En este punto, las bases de datos se comportan como "esquemas" de Oracle.
- Las bases de datos son independientes entre ellas, no se pueden ver objetos de una base de datos desde otra base de datos, excepto si se usan *dblinks.* Esto marca cómo se deben crear las bases de datos:
    - si es un servidor que acoge proyectos separados y no se han de interrelacionar: separación en bases de datos distintas.
    - si es un servidor de proyectos con recursos comunes: una única base de datos y distintos esquemas para cada proyecto.

5.3.2 Jerarquía de Objetos a nivel lógico:

Veamos aquí una breve descripción de los objetos tal como salen en el pgAdmin3:

- Servidores
    - Bases de datos (template0, template1 y postgres)
        - Cast: conversiones entre tipos de datos
        - Lenguajes
        - Esquemas
            - Agregados: definir funciones de agregación
            - Conversiones: definir nuevas conversiones de codificación de caracteres
            - Dominios: crear dominios que se pueden usar en la definición de tablas
            - Funciones
            - Funciones disparadoras
            - Procedimientos
            - Operadores
            - Operador de clases
            - Secuencias
            - Tablas
            - Tipos
            - Vistas
        - Replicación
    - Tablespaces
    - Roles de grupo (roles)
    - Roles de login (usuarios)

5.3.3 Creacion de tablespaces

Si queremos especificar ubicaciones alternativas para determinadas bases de datos o tablas, por ejemplo, queremos que ciertas tablas estén en otros discos distintos a los que se encuentran por defecto, debemos crear tablespaces.

```
CREATE TABLESPACE tablespacename [ OWNER username ] LOCATION 'directory'
```

- *tablespacename*: no puede empezar en “pg_” porque está reservado a los tablespaces de sistema.
- *username*: debe ser un superusuario del cluster.
- *directory:* se deben indicar trayectorias absolutas y el propietario del directorio debe ser el usuario con el que se ha instalado PostgreSQL.

En versiones anteriores de PostgreSQL, existía el comando “*initlocation”*, que servía para indicar ubicaciones alternativas para las bases de datos, y que exigía que se definiese una variable de entorno similar a *PGDATA* para cada base de datos.

Nota: cuando se crean tablespaces, en *$PGDATA/pg_tblspc* se crean enlaces simbólicos a los directorios físicos donde se encuentran, esto es importante tenerlo en cuenta en procesos como el cambio de ubicaciones o en la restauración de ficheros desde una copia de seguridad.

5.3.4 Creación de bases de datos

Una base de datos se puede crear desde la línea de comandos del sistema operativo (con el usuario de sistema) o desde una conexión a una base de datos.(con un usuario que tenga privilegios para crear bases de datos).

Desde el sistema operativo:

```
$ createdb [OPCIÓN]... [NOMBRE] [DESCRIPCIÓN]

Opciones:
  -D, --tablespace=TBLSPC   tablespace por omisión de la base de datos
  -E, --encoding=CODIFICACIÓN
                            codificación para la base de datos
  -O, --owner=DUEÑO         usuario que será dueño de la base de datos
  -T, --template=PATRÓN     base de datos patrón a copiar
  -e, --echo                mostrar los comandos enviados al servidor
  -q, --quiet               no desplegar mensajes
  --help                    mostrar esta ayuda y salir
  --version                 mostrar el n·mero de versión y salir

Opciones de conexión:
  -h, --host=ANFITRIÓN      nombre del servidor o directorio del socket
  -p, --port=PUERTO         puerto del servidor
  -U, --username=USUARIO    nombre de usuario para la conexión
  -W, --password            preguntar la contraseña

Si no se especifica, se creará una base de datos con el mismo nombre que el
usuario actual.
```

como se ve, desde el sistema operativo podemos crear bases de datos en otros servidores.

Desde un cliente SQL:

```
CREATE DATABASE name
    [ [ WITH ] [ OWNER [=] dbowner ]
            [ TEMPLATE [=] template ]
            [ ENCODING [=] encoding ]
            [ TABLESPACE [=] tablespace ]
            [ CONNECTION LIMIT [=] connlimit ] ]
```

connlimit marca el número de conexiones concurrentes que pueden haber a la base de datos, tiene por defecto el valor "-1", que quiere decir que no hay límite de conexiones.

5.3.5 Creación de roles (usuarios)

En PostgreSQL los usuarios son tipos de roles, el *role* es el concepto general. Esto es así a partir de PostgreSQL 8, porque en versiones anteriores no existían los roles, solo *usuarios* y *grupos,* a semejanza de los grupos de usuarios de los sistemas operativos.

Los roles son globales al cluster, se almacenan en la tabla del catálogo *pg_authid* y se pueden consultar en las vistas *pg_user y pg_roles.*

Desde el sistema operativo:

```
  createuser [OPCIÓN]... [ROL]

Opciones:
  -s, --superuser           el rol será un superusuario
  -S, --no-superuser        el rol no será un superusuario
  -d, --createdb            el rol podrá crear bases de datos
  -D, --no-createdb         el rol no podrá crear bases de datos
  -r, --createrole          el rol podrá crear otros roles
  -R, --no-createrole       el rol no podrá crear otros roles
  -l, --login               el rol podrá conectarse (predeterminado)
  -L, --no-login            el rol no podrá conectarse
  -i, --inherit             el rol heredará los privilegios de los roles de
                            los cuales es miembro (predeterminado)
  -I, --no-inherit          rol no heredará privilegios
  -c, --connection-limit=N  límite de conexiones para el rol
                            (predeterminado: sin límite)
  -P, --pwprompt            asignar una contraseña al nuevo rol
  -E, --encrypted           almacenar la constraseña cifrada
  -N, --unencrypted         almacenar la contraseña sin cifrar
  -e, --echo                mostrar los comandos a medida que se ejecutan
  -q, --quiet               no escribir ningún mensaje
  --help                    desplegar esta ayuda y salir
  --version                 desplegar información de versión y salir

Opciones de conexión:
  -h, --host=ANFITRIÓN      nombre del servidor o directorio del socket
  -p, --port=PUERTO         puerto del servidor
  -U, --username=NOMBRE     nombre de usuario con el cual conectarse
                            (no el usuario a crear)
  -W, --password            pedir contraseña para conectarse

Si no se especifican -s, -S, -d, -D, -r, -R o el ROL, se preguntará
interactivamente.
```

Desde un cliente SQL:

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]

donde option puede ser:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | CREATEUSER | NOCREATEUSER
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'timestamp'
    | IN ROLE rolename [, ...]
    | IN GROUP rolename [, ...]
    | ROLE rolename [, ...]
    | ADMIN rolename [, ...]
    | USER rolename [, ...]
    | SYSID uid
```

en cursiva los valores por defecto y algunas opciones están obsoletas pero se mantienen por compatibilidad con versiones anteriores.

El comando *CREATE USER* es ahora un alias de CREATE ROLE, pero con la opción LOGIN por defecto.

- | IN ROLE *rolename* [, ...]: añade el role nuevo (name) como miembro de los rolenames listados, equivale a hacer

  ```
  GRANT rolename1, rolename2 TO name;
  ```

- | IN GROUP *rolename* [, ...] ó | ROLE *rolename* [, ...]: la lista de rolename es añadido como miembro de name, equivale a hacer:

  ```
  GRANT name TO rolename1, rolename2;
  ```

5.3.6. Esquemas

Por último, las bases de datos se organizan mediante *esquemas*, contenedores lógicos de objetos de base de datos (tablas, vistas, procedimientos, etc.), básicamente son un espacio de nombres. Es característico de los esquemas:

- tienen un propietario.
- Permiten el uso de la base de datos por múltiples usuarios sin interferencias.
- Permiten que se puedan instalar aplicaciones realizadas por terceros si que existan colisiones en los nombres de los objetos.

```
CREATE SCHEMA schemaname [ AUTHORIZATION username ] [ schema_element [ ... ] ];
```

- `AUTHORIZATION username` : si se omite, el propietario del esquema es el usuario que ejecuta la sentencia.

- *`schema_element`* [...]: permite sentencias DDL dentro de la creación del esquema.

Para poder ver objetos de un esquema, debe usarse la notación

esquema.objeto

ya que en PostgreSQL no existe el concepto de sinónimos como en Oracle.

Existe una variable, que se puede definir a nivel de cluster o a nivel de cada usuario, llamada *SEARCH_PATH* donde se puede indicar una lista de esquemas en los que se buscarán objetos en caso de que no se ponga el nombre del propietario. En caso de que hayan nombres duplicados entre distintos esquemas que estén en esta lista, se cogerá el primero encontrado.

Capítulo 6. Instalación, desinstalación y migración.

PostgreSQL se puede instalar en muchas plataformas, las más conocidas Windows y Linux, en este curso vamos a ver cómo instalarlo en Linux. Además, la instalación se puede hacer desde los paquetes fuente del producto o bien desde paquetes o programas de instalación propios de cada distribución de software (este es el método más sencillo).

En http://www.postgresql.org disponemos de los ficheros de instalación, así como en las distintas distribuciones de Linux puede encontrarse disponible.

6.1 Instalación en Windows

La instalación en Windows es muy sencilla, consiste en bajarse un paquete comprimido, ejecutar el setup e ir respondiendo a las opciones que nos piden. Entre estas opciones están la creación de un usuario de Windows para lanzar los servicios y otro usuario para superusuario de la base de datos.

En un determinado momento nos preguntan los módulos que queremos cargar, podemos elegir estas opciones:

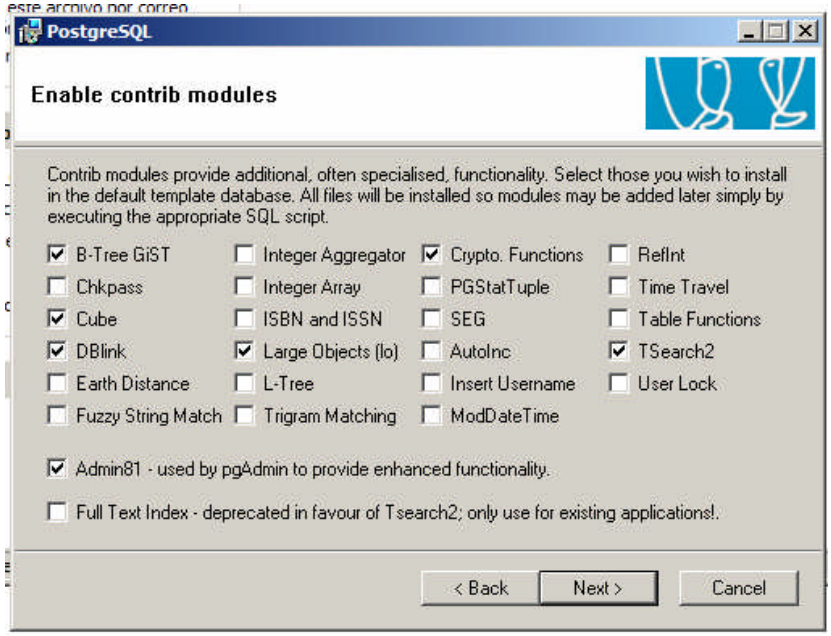

Figura 6-1: Opciones de instalación en Windows

En Windows sólo instala soporte para el lenguaje Plpgsql. Cuando finaliza la instalación, se ha creado un grupo de programas que nos permite iniciar o parar la base de datos, ir a los ficheros de configuración, ejecutar una consola con *psql*, o ejecutar *pgadmin3* (viene con el paquete de PostgreSQL en Windows).

Si se gasta la consola de *psql*, podemos tener problemas con los códigos de páginas, ya que normalmente Windows funciona con la página 850 y PostgreSQL quiere la 1252 si lo instalamos con LATIN1. Para resolver estos problemas ver:

http://www.postgresql.org/docs/current/static/install-win32.html

6.2 Instalación en Linux desde paquetes

La instalación a partir de de paquetes de una distribución determinada tiene sus ventajas e inconvenientes:

Ventajas:

- Facilidad de implantación, Los ficheros van a directorios predeterminados. Es un buen método para tener instalado PostgreSQL con poco esfuerzo y ponerse a practicar.
- Se crea automáticamente toda la infraestructura de funcionamiento, el usuario del sistema operativo, los programas de arranque y parada en el *init.d*, etc.
- Crea una instancia de base de datos, *cluster.*
- Incluso en algunas distribuciones , viene como una opción más que se puede instalar.

Inconvenientes:

- Poca flexibilidad, no podemos elegir en qué directorios instalar, aunque suele ser suficiente
- No suelen venir las últimas versiones en el caso de que vengan suministrados por la distribución, aunque probablemente en la web de PostgreSQL o en la de la distribución ya tengan el paquete preparado para la última versión.

Instalación / Desinstalación a partir de RPM's:

Se puede instalar el paquete básico y luego los paquetes opcionales, o instalar el paquete completo:

```
$ rpm –Uvh postgresql-version.rpm
```

Para desinstalarlo, hay que parar el servidor y:

```
$ rpm –e postgresql-*
      (en órden correcto si hemos instalado el paquete básico y opciones, por las dependencias)
$ rm – r /var/lib/pgsql
$ userdel postgres
```

Instalación / Desinstalación con apt-get (distribuciones Debian):

Se puede instalar el paquete básico y luego los paquetes opcionales, o instalar el paquete completo:

```
$ apt-get install postgresql
      si faltan dependencias podemos hacer
$apt-get build-dep postgresql-version
```

Para desinstalarlo, hay que parar el servidor y:

```
$ aptitude purge postgresql-version --purge-unused
```

6.3 Instalación en Linux desde los ficheros fuente

Este método tiene como inconveniente la dificultad de implantación, ya que requiere que el instalador se plantee una serie de requerimientos, así como se deben realizar operaciones que requieren estar famililiarizados con el sistema operativo, aunque en la documentación explican paso a paso todo lo que hay que hacer.

Las ventajas, en cambio, son muchas:

- instalación controlada totalmente
- fijamos la ubicación de todos los directorios y ficheros, tanto los del programa como los de los datos.
- podemos instalar características opcionales: soporte extendido para tipos de datos, soporte nativo de idiomas, etc.
- podemos instalar paquetes opcionales: tcl/tk, python, java, perl, PAM, OpenSSL, etc.
- total flexibilidad para configurar a gusto del administrador.

A continuación se detalla todo el proceso de instalación.

6.3.1 Decisiones iniciales, configuración del entorno

Antes de empezar la instalación, conviene plantearse algunas cosas:

- usuario de sistema de PostgreSQL, normalmente se crea un usuario llamado “postgres”, que es el que se usará para las tareas propias de mantenimiento
- usuario con el que vamos a instalar, porque puede instalarlo todo root y luego darle ciertos permisos al usuario “postgres”. Lo mejor es realizar ciertas operaciones de instalación con el usuario postgres y otras con root.
- determinar los directorios donde se va a realizar la instalación de PostgreSQL así como los directorios donde se crearan los cluster (aunque esta decisión no hace falta tomarla ahora). De este modo, se pueden definir unas variables de entorno que nos faciliten el trabajo.

Con el usuario root creamos el usuario postgres:

```
$ adduser –-home /home/postgres postgres
```

Luego podemos configurar una serie de variables de entorno para este usuario, lo cual nos facilitará el trabajo de instalación y mantenimiento:

```
$ su – postgres
$ vi .bash_profile     -- o el fichero equivalente: .profile, .bashrc …

recomiendo estas variables, los nombres son sugerencias, y ninguna es
estrictamente necesaria, aunque las vamos a utilizar a lo largo del manual:

# ubicación cluster, mejor un directorio que esté en otro disco
export PGDATA=$HOME/data

# fichero de log del servidor
export PGLOG=$PGDATA/server.log
```

```
# directorio desde donde realizaremos la instalación, conviene conservarlo
export PGSRC=$HOME/instalar/postgresql-8.2.5
# ruta donde se instala PostgreSQL
export PGHOME=$HOME/pgsql
# ruta a los programas de PostgreSQL
export PGBIN=$HOME/pgsql/bin
```

6.3.2 Descarga del software

Vamos a instalar la versión 8.2.5 (última disponible en el momento de la elaboración del manual).

Buscamos el código fuente en www.postgresql.org y lo descargamos, por ejemplo en el directorio `$HOME/instalar`. Si no tenemos entorno gráfico, podemos hacerlo usando wget:

```
$ mkdir $HOME/instalar
$ cd $HOME/instalar
$ wget ftp://ftp.postgresql.org/pub/source/v8.2.5/postgresql-8.2.5.tar.gz
$ tar xvfz postgresql-8.2.5.tar.gz
$ cd postgresql-8.2.5     -- este es el directorio $PGSRC
```

Nota: $PGSRC puede estar ubicado en un directorio temporal, y puede ser borrado al terminar la instalación, el problema viene si posteriormente queremos reinstalar, o añadir módulos, deberemos volver a configurar la instalación y repetir muchos pasos. Dado que no ocupa mucho espacio, yo recomiendo no borrar estos directorios.

Una vez descargado, disponemos de unas instrucciones de instalación en $PGSRC/INSTALL, en este fichero nos explica incluso cómo hacer una instalación rápida.

6.3.3 Requerimientos y configuración de instalación

Dado que instalamos a partir de ficheros fuentes, necesitamos asegurarnos que tenemos las herramientas necesarias:

- comprobar que *el gcc* está.
- comprobar que disponemos del programa *make o gmake.*
- variables de entorno que tienen influencia: CC (compilador), CFLAGS (compiler flags), LDFLAGS (linker flags), CPPFLAGS (preprocessor flags), CPP (preprocessor).

Además, se deben cumplir una serie de requerimientos de configuración del kernel y de configuración de hardware.

También, en función de las opciones que elijamos, necesitaremos que estén instaladas ciertas librerías o módulos, por ejemplo:

- gettext: para manejo de cadenas
- readline: para el buffer de órdenes de psql
- zlib: funciones necesarias para los programas de backup y empaquetado

- openSSL si queremos que las conexiones sean mediante SSL

Ejecutamos el programa que configura la instalación:

```
$ $PGSRC/configure --help
```

este programa comprueba si se cumplen los requerimientos para realizar la instalación que se elija, en caso de que no sea así, devolverá un mensaje de error y no podremos continuar hasta que corrijamos el problema.

Si ejecutamos:

```
$ $PGSRC/configure –config-cache
```

se guardan las opciones de instalación y se permiten realizar actualizaciones acumulativas fácilmente.

Instalamos, por ejemplo, con las siguientes opciones:

```
./configure \
--prefix=/home/postgres/pgsql \
--with-cXX \
--with-maxbackends=100 \
--with-perl \
--with-plpgsql \
--with-pgport=5432 \
--enable-nls \
--enable-locale
```

6.3.4 Compilación y enlazado

con el usuario postgres:

```
$ make

o tambien:

$ make clean && make
```

y luego con el usuario root:

```
$ make install
```

ejecutando **pg_config** podemos ver cómo ha sido instalado PostgreSQL.

6.3.5 Configurar el usuario postgres y otros usuarios

Aparte de la configuración de entorno recomendada anteriormente, aquí hay que realizar una serie de configuraciones obligadas para el usuario postgres o para cualquiera que desde la máquina quiera usar o conectarse a PostgreSQL:

- PATH: incluir $PGBIN en el path
- MANPATH: incluir el directorio de los manuales
- LD_LIBRARY_PATH: incluir el directorio de librerías

y a título informativo, se pueden crear estas otras variables:

- PGHOST: host servidor de PostgreSQL.
- PGPORT: puerto donde escucha el postmaster del servidor, 5432 por defecto

```
export PATH=$PGBIN:$PATH
export MANPATH=$PGHOME/man:$MANPATH
export LD_LIBRARY_PATH=$PGHOME/lib:$LD_LIBRARY_PATH

export PGHOST=miservidor
export PGPORT=5432
```

6.3.6 Desinstalación:

```
con el usuario root:

$ cd $PGSRC
$ make uninstall
$ rm –r $PGHOME
$ rm –r /home/postgres
$ deluser postgres
```

6.3.7 Instalación de un cliente de PostgreSQL:

En ocasiones, no queremos instalar la base de datos, si no el software mínimo para conectarnos a otras bases de datos.

Si se trata de un cliente Windows, basta con instalar PgAdmin III, que ya incluye un cliente.

Si se trata de un cliente Postgres, si es un PC de usuario, probablemente el Yast lo tenga entre sus opciones de instalación y si es un servidor, se debe realizar lo siguiente, en vez de

```
$ make install
```

ejecutar

```
$ make -C src/bin install
$ make -C src/include install
$ make -C src/interfaces install
$ make -C doc install
```

6.4 Migración (de una versión a otra)

6.4.1 Si no queremos preservar el cluster

Si no queremos preservar el cluster, es decir, los usuarios y los datos, es muy fácil, se trata de:

- parar el proceso *postmaster*
- salir del usuario *postgres*
- desinstalar
- borrar el cluster
- instalar la nueva versión
- crear el cluster

6.4.2 Si queremos preservar el cluster

Aquí depende de la versión inicial y de la versión final:

Migración de una versión X.Y.Z a X.Y.W (tercer dígito):

el cluster es normalmente compatible, no obstante, conviene consultar en la documentación de instalación.

- parar el postmaster
- salir del usuario *postgres*
- desinstalar el sofware
- copiar el directorio donde está el cluster a otro (copia de seguridad)
- instalar la nueva versión
- copiar la copia del cluster otra vez al origen

Migración de una versión X.Y.Z a X.V.W (primero o segundo dígito):

El cluster no es normalmente compatible. Suponiendo que PostgreSQL esté instalado en */home/postgres/pgsqls* seguiríamos estos pasos aproximadamente:

- con el *postmaster* en marcha
- hacemos un volcado del cluster ($ pg_dumpall > /tmp/cluster.dat)
- parar el *postmaster*

- salir del usuario *postgres*
- mover /*home/postgres/pgsql a /home/postgres/pgsql.old* (mejor que borrar)
- desinstalar el sofware
- instalar la nueva versión
- crear un nuevo cluster con la nuev versión
- arrancar el servidor
- $ psql template1 < /tmp/cluster.dat
- restaurar alguna configuración de */home/postgres/pgsql,old.*
- si todo va bien, borrar */home/postgres/pgsql,old.*

Nota: hay que tener cuidado con los tablespaces, ya que usan enlaces simbólicos y pueden quedar incorrectos.

6.5 Instalación de dos versiones en la misma máquina

Aunque no es recomendable, en ocasiones deben convivir dos versiones de PostgreSQL en la misma máquina, esto ocurre sobre todo en entornos de producción.

- Cada versión puede pertenecer a un usuario de sistema operativo distinto o pueden ser el mismo
- cada versión en un directorio distinto, aquí se podrían usar enlaces simbólicos que apunten a los programas y datos más recientes.
- hay que ejecutar los postmaster en distintos puertos

6.6 Gestión de los recursos del Kernel

La instalación de PostgreSQL requiere la comprobación de que el servidor será capaz de soportarlo. Normalmente, los valores por defecto son más que suficientes. Estos recusos afectan a la **memoria compartida** y a los **semáforos**. Será tarea del administrador de sistemas cambiar los valores de los parámetros si es necesario. Si el sistema no puede proporcionar los recursos que requierel servidor, éste no se puede poner en marcha y devuelve un error. Para una instalación normal, los valores por defecto son suficientes.

6.6.1 Parámetros de PostgreSQL que afectan a la memoria compartida

los valores de estos parámetros están en *$PGDATA/postgresql.conf:*

Parámetro	Descripción	Bytes
max_locks_per_transaction	• cada uno consume aproximadamente 220 bytes • valor = 64	12.800
max_connections	• cada una consume aproximadamente 400 bytes + 220* *max_locks_per_transaction*. • valor = 100 (al decir –*with-backends*=100)	1.320.000
max_prepared_transactions	• cada una consume aproximadamente 600 bytes + 220* *max_locks_per_transaction*. • valor = 5	67.000
shared_buffers	• cada buffer es de 8Kb (asumiendo bloques de BD de 8Kb) • se recomienda como mínimo *max_connections* 2* • un valor biueno es 8.000	65.536.000
wal_buffesr	• cada uno consume 8Kb • valor = 8 • por ahí recomiendan 8200	67.174.400
max_fsm_relations	• cada una consume 70 bytes • valor = 1000	70.000
max_fsm_pages	• cada una consume 6 bytes • recomiendan max_fsm_relations*16 • valor = 20.000	120.000

6.6.2 Parámetros del Kernel:

Parámetro	Descripción	Valores razonables
SHMMAX	tamaño máximo del segmento de memoria compartida	varios Mb (pueden ser decenas o centenares)
SHMMIN	tamaño mínimo del segmento de memoria compartida	500 Kb
SHMALL	cantidad total de memoria compartida disponible	si se da en bytes como SHMMAX, si en páginas SHMMAX / PAGE_SIZE
SHMSEG	número máximo de segmentos de memoria compartida por proceso	1 (el valor por defecto es mayor)
SHMNI	número máximo de segmentos de memoria compartida	SHMSEG + los que requieran otras aplicaciones
SEMMNI	número máximo de identificadores de semáforos	al menos max_connection / 16
SEMMNS	número máximo de semáforos del sistema	(max_connection / 16)*17 + los que requieran otras aplicaciones
SEMMSL	número máximo de semáforos del conjunto	al menos 17
SEMMAP	número de entradas en un mapa de semáforos	incrementarlo como SEMMNS
SEMVMX	valor máximo de semáforo	al menos 1000 (por defecto 32767)

6.6.3 Memory Overcommit

La gestión de Linux 2.4 y posteriores de la memoria virtual no es óptima para PostgreSQL porque a veces el kernel mata el postmaster si otro proceso demanda memoria y el sistema se queda sin memoria virtual.

La solución es o poner servidores dedicados, o con suficiente memoria, o bien deshabilitar esta opción del sistema:

```
$ sysctl –w vm.overcommit_memory=2
```

Capítulo 7. Puesta en marcha

Una vez instalado el software estamos en disposición de empezar a usar este SGBD, pero no tenemos todavía creado ningún cluster de base de datos

7.1 Creacion del cluster

Ya vimos en un punto anterior cómo se crea un cluster de base de datos, por ejemplo, en este curso, vamos a crear el siguiente cluster:

```
$ initdb --pgdata=$PGDATA --encoding=LATIN1 --locale=es_ES
```

Con este proceso se han creado tres bases de datos: *template0* (que ya se ha dicho que no se debe modificar ni debemos conectarnos a ella*), template1 y postgres.*

Al terminar de crear el cluster nos responde que ya podemos iniciar el postmaster de este cluster de dos modos:

```
$ postmaster -D $PGDATA
```

o

```
$ pg_ctl -D $PGDATA -l $PGLOG start
```

Lo hacemos del segundo modo, ahora, una vez arrancada la instancia, podemos conectarnos desde nuestro servidor a la base de datos usando el *psql*:

```
$ psql

    podemos cambiar la clave al superusuario:
# alter user postgres password ‘postgres’;
# \q
```

pero no podremos conectarnos desde otros ordenadores, con lo que no podemos ejecutar el pgadmin3 contra esta base de datos, más adelante haremos la configuración adecuada.

Si queremos parar el servidor:

```
$ pg_ctl stop
```

7.2 Modificacion de template1 + añadir extensiones-contrib

Hay que tener en cuenta ciertos aspectos antes de empezar a trabajar con nuestras bases de datos
:

- *template1* es una plantilla para crear otras bases de datos, conviene que, antes de crear nuevas bases de datos, la completemos con todos aquellos elementos que consideremos que deben estar en todas las bases de datos que creemos, lenguajes, módulos opcionales (*tsearch2, lo, dblink, etc*), tablas comunes, etc. Más adelante veremos cómo se instalan estas opciones.
- ¿va a ser *postgres* nuestra base de datos o la reservamos para pruebas o como plantilla y no usamos *template1*?, ¿vamos a crear más bases de datos?

Para modificar *template1* no hay más que conectarse a ella y ejecutar los scripts de base de datos que hagan falta.

Por ejemplo, las bases de datos se suelen crear sin lenguajes, no existe el *plpgsql*, para instalarlo, ejecutamos desde el sistema operativo:

```
$ createlang plpgsql template1
$ createlang plpgsql postgres
```

lo ejecutamos también en postgres para que tenga las mismas opciones.

Pero además, en el mundo del software libre se crea código para añadir nuevas funcionalidades. Estos módulos los hemos añadido, por ejemplo, en la instalación de PostgreSQL en Windows, pero no en la de Linux, para poder instalarlos hay que ir al directorio *$PGSRC/contrib.*

En *$PGSRC/contrib* hay un fichero README que indica el uso de cada una de estas opciones.

Ejemplos de opciones que se tienen que instalar según los requerimientos de las aplicaciones:

- *tsearch2:* soporte para indexación usando GiST
- *lo:* mantenimiento de tipos de datos LOB
- *vacuumlo:* vacuum para objetos LOB
- *dblink:* poder ejecutar sentencias en otras bases de datos
- *pgAdmin3:* funciones de instrumentación.

dentro de cada directorio hay un fichero con las instrucciones de instalación.

Como ejemplo, para instalar "*lo"* se siguen estos pasos, suponiendo que estamos conectados con el usuario postgres:

```
$ su – root
$ cd $PGSRC/contrib/lo
$ make install
$ exit
$ cd $PGSRC/contrib/lo
$ psql –U postgres template1 < lo.sql
$ psql –U postgres postgres < lo.sql
```

lo mismo hay que hacer para instalar "*tsearch2", cambiando "lo" por "tsearch2".* Para ver el efecto de instalar esta opción, ver que en el esquema *public* de las bases de datos hay 4 tablas que antes de instalar *tsearch2* no estaban (entre otras cosas).

Por último, se pueden instalar las *"extended features" de pgAdmin3* tal como recomienda este programa cuando arranca y detecta que no están instaladas en el servidor.

7.3 Puesta en marcha y parada del servidor

La puesta en marcha del servidor se puede hacer de forma manual con dos comandos:

- ***pg_ctl***: facilidad para la puesta en marcha, parada y reconfiguración del servidor. Hace uso de la instrucción que actúa sobre el servidor postgres.
- ***postmaster***: proceso servidor de base de datos.

El comando *pg_ctl* es el que se usa normalmente, porque permite controlar más aspectos que el comando *postmaster.*

7.3.1 Puesta en marcha usando *postmaster*

```
$ postmaster [OPCION]...

Opciones:

  -B NBUFFERS     número de búfers de memoria compartida
  -c VAR=VALOR    definir parámetro de ejecución
  -d 1-5          nivel de depuración
  -D DATADIR      directorio de bases de datos
  -F              desactivar fsync
  -h NOMBRE       nombre de host o dirección IP en que escuchar
  -i              activar conexiones TCP/IP
  -k DIRECTORIO   ubicación del socket Unix
  -l              activar conexiones SSL
  -N MAX-CONN     número máximo de conexiones permitidas
  -o OPCIONES     pasar «OPCIONES» a cada proceso servidor
  -p PUERTO       número de puerto en el cual escuchar
  -S              modo silencioso (en segundo plano sin salida de depuración)
  --help          desplegar esta ayuda y salir
  --version       desplegar número de versión y salir

Opciones de desarrollador:

  -n              no reinicializar memoria compartida después de salida anormal
  -s              enviar SIGSTOP a todos los backends si uno de ellos muere
```

las opciones de este comando se pueden fijar en el fichero de configuración *postgresql.conf,* se comentarán entonces.

7.3.2 Puesta en marcha y parada usando *pg_ctl*

pg_ctl es un programa para iniciar, detener, reiniciar, recargar archivos de configuración, reportar el estado de un servidor PostgreSQL o enviar una señal a un proceso PostgreSQL.

```
Empleo:

  pg_ctl start   [-w] [-D DATADIR] [-s] [-l ARCHIVO] [-o «OPCIONES»]
  pg_ctl stop    [-W] [-D DATADIR] [-s] [-m MODO-DETENCIÓN]
  pg_ctl restart [-w] [-D DATADIR] [-s] [-m MODO-DETENCIÓN] [-o «OPCIONES»]
  pg_ctl reload  [-D DATADIR] [-s]
  pg_ctl status  [-D DATADIR]
  pg_ctl kill    NOMBRE-SEÑAL ID-DE-PROCESO
  pg_ctl register   [-N SERVICIO] [-U USUARIO] [-P PASSWORD] [-D DATADIR]
                    [-w] [-o «OPCIONES»]
  pg_ctl unregister [-N SERVICIO]

Opciones comunes:
  -D, --pgdata DATADIR   ubicación del área de almacenamiento de datos
  -s, --silent           mostrar sólo errores, no mensajes de información
  -w                     esperar hasta que la operación se haya completado
  -W                     no esperar hasta que la operación se haya completado
  --help                 mostrar este texto y salir
  --version              mostrar información sobre versión y salir
(Por omisión se espera para las detenciones, pero no los inicios o reinicios)

Si la opción -D es omitida, se usa la variable de ambiente PGDATA.

Opciones para inicio y reinicio:
  -l  --log ARCHIVO      guardar el registro del servidor en ARCHIVO.
  -o OPCIONES            parámetros de línea de órdenes a pasar a postmaster
                         (ejecutable del servidor de PostgreSQL)
  -p RUTA-A-POSTMASTER   normalmente no es necesario

Opciones para detención y reinicio:
  -m MODO-DE-DETENCIÓN   puede ser «smart», «fast» o «immediate»

Modos de detención son:
  smart       salir después que todos los clientes se hayan desconectado
  fast        salir directamente, con apagado apropiado
  immediate   salir sin apagado completo; se ejecutará recuperación
              en el próximo inicio


Nombres de señales permitidos para kill:
  HUP INT QUIT ABRT TERM USR1 USR2

Opciones para registrar y dar de baja:
  -N SERVICIO            nombre de servicio con el cual registrar
                         el servidor PostgreSQL
  -P CONTRASEÑA          contraseña de la cuenta con la cual registrar
                         el servidor PostgreSQL
  -U USUARIO             nombre de usuario de la cuenta con la cual
                         registrar el servidor PostgreSQL
```

Operaciones:

- **start:** puesta en marcha del servidor
- **stop:** parada del servidor
- **restart:** parada del servidor seguida de puesta en marcha
- **reload:** envía al postmaster una señal SIGHUP, que provoca que recargue toda la información de configuración. Algunas opciones de configuración requieren parar el servidor.
- **status:** comprueba si hay un postmaster en marcha y muestra el PID y sus opciones de puesta en marcha.
- **kill, register, unregister:** opciones para la versión de Windows para matar, registrar como servicio o eliminar el servicio.

Puesta en marcha

```
$ pg_ctl start [-w] [-D DATADIR] [-s] [-l ARCHIVO] [-o «OPCIONES»]
```

por ejemplo:

```
$ pc_ctl –D $PGDATA –l $PGLOG`date +%j.%H.%M` start
```

arrancaría el servidor y además usaría un fichero de seguimiento con nombre variable en función del dia del año, la hora del día y los minutos.

Hay que tener presente que el proceso puede acabar bien y no arrancar, por ejemplo, si los permisos del directorio que se pone para el fichero de seguimiento no son correctos, por eso conviene comprobar siempre con

```
$ pg_ctl status
```

aunque la instrucción sólo mira si en $PGDATA está el *postmaster.pid.* Se puede borrar este fichero y el status será apagado, pero no dejará arrancar (el socket en /tmp/.s.PGSQL.$PGPORT está bloqueado) ni hacer stop (dirá que no está en marcha), en este caso hay que matar el proceso. Si queremos asegurarnos de si está arrancado, podemos ejecutar:

```
$ ps –ef | grep postgres
```

Parada manual

```
$ pg_ctl stop    [-W] [-D DATADIR] [-s] [-m s[mart] f[ast] i[mmediate]]
```

-m indica la forma de parar el servidor:

- ***s[mart]:*** espera que todos los clientes se desconecten y evita que se produzcan nuevas conexiones. Es la forma recomendable.
- ***f[ast]:*** no espera a que los clientes se desconecten, todas las transacciones activas son deshechas, luego los clientes son desconectados y se apaga la base de datos.

- ***i[mmediate]:*** aborta todos los procesos servidores sin una parada limpia. Será necesaria una recuperación cuando se levante la base de datos la siguente vez.

Puesta en marcha y parada automática

Como en todos los servidores, interesa que el servidor de PostgreSQL arranque y se pare cuando el servidor arranque y se pare respectivamente. Estas operaciones se hacen usando los scripts de arranque y parada del servidor, operaciones que realiza el usuario *root.*

Debe existir un fichero llamado */etc/init.d/postgresql.* Si no existe hay que copiarlo:

```
$ cp $PGSRC/contrib/start-scripts/linux /etc/init.d/postgresql
$ chmod u+x /etc/init.d/postgresql
```

Para que lo añada a los distintos niveles de ejecución, se deben crear los enlaces simbólicos en los niveles que se desee

```
$ ln –s /etc/init.d/postgresql /etc/rc3.d/S12postgresql
$ ln –s /etc/init.d/postgresql /etc/rc3.d/K02postgresql
```

o bien, usar las utilidades que tengan en cada distribución para ello, por ejemplo:

```
$ cd /etc/init.d

# en Suse o RedHat:
$ /sbin/chkconfig –add postgresql
$ /sbin/chkconfig postgresql on

# en Debian:
$ update-rc.d [-n] postgresql defaults 98 02
```

Donde los niveles de arranque eran:

0	parada
1	monousuario (mantenimiento)
2	usos varios
3	multiusuario, red disponible
4	usos varios
5	servidor X window
6	reiniciar (shutdown/reboot)

Capítulo 8. Configuración del entorno de ejecución

Existe un conjunto de parámetros que determinan el entorno de ejecución, todos se encuentran en el fichero *$PGDATA/postgresql.conf*, aunque algunos de ellos se pueden fijar mediante las opciones de arranque del servidor PostgreSQL.

Los parámetros tienen valores por defecto:

- vinculados al código fuente, bien como un valor simbólico en */include/pg_config.h* o como una opción de configuración en tiempo de compilación.
- por ser una opción de puesta en marcha del servidor en el comano *postmaster*

Los valores de los parámetros se pueden cambiar:

- con el servidor en marcha con el comando SET
- cambiando el fichero y recargando la configuración con *pg_ctl reload,* aunque hay parámetros que requieren parar y arrancar.

8.1 Ubicación de ficheros

parámetro	uso	modificación
data_directory = 'ConfigDir'	ConfigDir representa $PGDATA	puesta en marcha del servidor
config_file	nombre fichero configuración	en línea de comandos de postmaster
hba_file = 'ConfigDir/pg_hba.conf'	fichero de configuración de autenticación	puesta en marcha del servidor
ident_file = 'ConfigDir/pg_ident.conf'	fichero de configuración de la autenticación	puesta en marcha del servidor
external_pid_file = '(none)'	nombre del fichero pid	puesta en marcha del servidor

8.2 Conexión

parámetro	uso	modificación
listen_adresses= 'localhost'	especifica las direcciones IP que el servidor debe escuchar desde aplicaciones cliente. Lista separada por comas, si está vacía no acepta conexiones por red, si es '*' acepta cualquier dirección ('localhost')	puesta en marcha del servidor. Conviene poner la IP o nombre con dominio del servidor para poder recibir conexiones de clientes.
port = 5432	puerto TCP/IP donde escucha postmaster (5432)	puesta en marcha del servidor o con *postmaster –p puerto*
max_connections = 1000	máximo de sesiones concurrentes (100)	puesta en marcha del servidor o con *postmaster –n numero*
superuser_reserved_connections = 2	conexiones reservadas para superusuarios (2)	puesta en marcha del servidor
unix_socket_directory = ''	indica donde se encuentran los socket del dominio local para las conexiones locales (/tmp)	puesta en marcha del servidor o *postmaster –k directorio*

parámetro	uso	modificación
unix_socket_group = ’’		
unix_socket_permissions = 0777	si ponemos 0700 solo dejamos conectar al usuario postgres	
bonjour_name = ‘’		puesta en marcha del servidor
tcp_keepalives_idle = 0 tcp_keepalives_interval = 0 tcp_keepalives_count = 0	controlar si las conexiones siguen vivas, para matarlas si no responden	

8.3 Seguridad y autenticado

parámetro	uso	modificación
authentication_timeout = 60	tiempo máximo en segundos para completar el autenticado del cliente	puesta en marcha del servidor
ssl = off	si el postmaster negocia con clientes que usen conexiones ssl	puesta en marcha del servidor *o postmaster –l*
password_encryption = on	cómo se almacena por defecto una palabra de paso (on, encriptada)	puesta en marcha del servidor
krb_server_keyfile, krb_svrname, krb_server_servername, krb_caseins_users	no se suele utilizar, configura el modo de usar el autenticado con Kerberos	

8.4 Uso de recursos

parámetro	uso	modificación
Memoria		
shared_buffers = 1000	número de buffers que se van a crear en la memoria compartida, determinando asi el tamaño de la caché de base de datos	puesta en marcha del servidor o *postmaster –B número*
temp_buffers = 1000	máximo número de buffers temporales que una sesión puede usar, se usan para acceder a las tablas temporales	se modifica en la sesión (SET …)
max_prepared_transactions = 5	número máximo de transacciones preparadas simultáneamente. Se recomienda que sean al menos *max_connections* si se desean usar	
work_mem = 1024	cantidad de memoria total en Kb que se usará para operaciones de ordenación antes de usar ficheros temporales en disco	
maintenance_work_mem = 16384	cantidad máxima de memoria en Kb que se usará para operaciones de mantenimiento	
max_stack_depth = 2048	profundidad máxima de seguridad de la pila de ejecución del servidor (ver *ulimit –s)*	puesta en marcha del servidor
Espacio libre		
max_fsm_relations = 1000	número máximo de tablas que el gestor de espacio libre (FSM) manipulará en la memoria compartida. Aproximadamente 70 bytes por cada relación	puesta en marcha del servidor
max_fsm_pages = 20000	con el anterior, determinan el tamaño del espacio libre en cache usado por el FSM. Afecta al rendimiento frente a inserciones y borrados	puesta en marcha del servidor

parámetro	uso	modificación
Uso de recursos del kernel		
max_files_per_process = 1000	nº máximo de ficheros abiertos por cada proceso servidor. Si aparece el error en el SO de "too many files" reducir este valor o cambiar el kernel	puesta en marcha del servidor
preload_libraries = ''	especifica las librerias compartidas que deben cargarse en la puesta en marcha del servidor	puesta en marcha del servidor
max_locks_per_transactions = 64	junto con *max_connections* determinan el tamaño de la tabla de bloqueos compartida	
Retraso del vacuum basado en coste en PostgreSQL 8 se introduce el *autovacuum*, y surgen problemas en la ejecución concurrente de *vacuum y analyze* con sesiones en marcha		
vacuum_cost_delay = 0	tiempo en ms que se duerme el proceso *vacuum* cuando se ha excedido el *vacuum_cost_limit*	puesta en marcha del servidor
vacuum_cost_page_hit = 1	coste estimado de limpiar un buffer de la cache de BD, bloquear la pila del buffer, actualizar la tabla compartida hash, y escanear el contenido de la página	puesta en marcha del servidor
vacuum_cost_page_miss = 10	coste estimado en limpiar un buffer que tiene que ser leido del disco, bloquear la pila del buffer, actualizar la tabla compartida hash, leer el bloque desde disco y escanear el contenido de la página	puesta en marcha del servidor
vacuum_cost_page_dirty = 20	coste estimado cuando la limpieza modifica un buffer limpio dejándolo sucio. Trabajo extra de llevar el buffer a disco	puesta en marcha del servidor
vacuum_cost_limit = 200	coste acumulado que hace que el vacuum se pare momentáneamente	puesta en marcha del servidor
proceso de fondo Writer proceso de fondo que se dedica a escribir los búferes sucios, los procesos servidores no esperan en sus escrituras, con lo que se mejora el rendimiento en el procesamiento de los puntos de verificación y se produce un pequeño incremento en el I/O porque un buffer se puede escribir varias veces.		
bgwriter_delay = 200	tiempo en ms entre cada puesta en marcha del writer	puesta en marcha del servidor
bgwriter_lru_percent = 1.0	porcentaje de bufferes que están próximos su reciclado y escribe los que están sucios	puesta en marcha del servidor
bgwriter_lru_maxpages = 5	nº de buferes que serán escritos por el método de estar próximo su reciclado	puesta en marcha del servidor
bgwriter_all_percent = 0.333	porcentaje máximo de todos los bufferes que serán examinados para ver si están sucios y se tienen que escribir	puesta en marcha del servidor
bgwriter_all_maxpages = 5	nº máximo de buferes que serán escritos en cada escaneado que efectúa writer para ver si están sucios y escribirlos	puesta en marcha del servidor

8.5 WAL (Write Ahead Log, mantenimiento del diario)

parámetro	uso	modificación
configuración del mantenimiento del diario		
fsync = true	determina si PostgreSQL debe forzar al SO a escribir en disco y que no utilice los mecanismos de buffering que tenga	puesta en marcha del servidor o recarga o *postmaster -F*

parámetro	uso	modificación
wal_sync_method = fsync	método que usa el gestor WAL para forzar la escritura de los buffers en disco. Posibles valores: fsync, fdatasunc, open_sync, open_datasync, fsync_writethrough (el valor por defecto depende de la plataforma, en linux en fsync)	recarga o puesta en marcha del servidor
full_pages_writes = on	establece si se copia el bloque entero al diario después de la primera modificación que sufra o no, después de un punto de verificación (por defecto está a on, para evitar problemas de corrupción de bloques).	
wal_buffers = 8	cantidad de buffers para los diarios que forman la cache de la WAL. A mayor tamaño, mayor rendimiento	puesta en marcha del servidor
commit_delay = 0	tiempo en microsegundos que espera el servidor para llevar las entradas del buffer de diario a los diarios después de que una transacción se confirme. Permite que otros procesos puedan aprovechar esta operación, es decir, que se haga un solo commit para varias transacciones.	por instrucción: *SET commit_delay TO espera*
commit_siblings = 5	nº mínimo de transacciones activas que debe de haber para que el gestor WAL se espere el tiempo indicado en *commit_delay*	por instrucción: *SET commit_siblings TO número*
puntos de verificación		
checkpoint_segments = 3	distancia máxima entre puntos de verificación automáticos, medida en nº de ficheros. Los diarios están divididos en segmentos de 16Mb, cuando se han llenado el nº de segmentos de este parámetro, se realiza un punto de verificación y así el sistema puede reutilizar los segmentos	recarga o puesta en marcha del servidor
checkpoint_timeout = 300	tiempo máximo en segundos entre la realización de un punto de verificación y el siguiente	recarga o puesta en marcha del servidor
checkpoint_warning = 30	escribe un mensaje en el fichero de seguimiento si un punto de verificación se lanza antes del intervalo dado en este parámetro, si es 0 se deshabilita el warning, en segundos	recarga o puesta en marcha del servidor
Archivamiento		
archive_command = ‘’	instrucción del SO que se debe ejecutar para archivar un diario completado WAL, si está vacío, el archivamiento automático está deshabilitado. Por ejemplo: `'cp "%p" /mnt/pg8/archivedir/"%f"'` donde %p es el camino completo al fichero y %f el nombre	recarga o puesta en marcha del servidor

8.6 Ajuste de rendimiento de consultas

parámetro	uso	modificación
Método de planificación y optimización		
enable_bitmapscan = on	habilita o deshabilita el uso de tipos de planes de escaneo en bitmaps	SET <nombre_parametro> TO [true | false]
enable_hashagg = on	habilita o deshabilita el uso de tipos de planes de agregación por dispersión	

parámetro	uso	modificación
enable_hashjoin = on	habilita o deshabilita el uso de tipos de planes concatenación	
enable_indexscan = on	habilita o deshabilita el uso de planes de recorrido en índice	
enable_mergejoin = on	habilita o deshabilita el uso de tipos de planes de concatenación por fusión	
enable_nestloop = on	habilita o deshabilita el uso de tipos de planes de concatenación de bucles anidados	
enable_seqscan = on	habilita o deshabilita el uso de tipos de planes de recorrido secuencial (full scan)	
enable_sort = on	habilita o deshabilita el uso de pasos de ordenación explícitos	
enable_tidscan = on	habilita o deshabilita el uso de planes del rastreo TID en el planificador. Se usa un rastreo TID cuando la pseudo-columna CTID aparece en un WHERE CTID es la localización física de una fila	
Constantes de coste		
effective_cache_size = 1000	tamaño efectivo de la cache de disco disponible para realizar un rastreo con índice (1000). Valor alto aumenta la probabilidad de rastreo con índice, en caso contrario, será secuencial	SET effective_cache_size TO tamaño
random_page_cost = 4	indica el coste de cargar una página al azar en cache (4). Se supone que el coste de carga de una página secuencial es 1	SET random_page_cost TO valor_decimal
cpu_tuple_cost = 0.01	indica el coste de procesar una tupla en una página de datos (0.01)	SET cpu_tuple_cost TO valor_decimal
cpu_index_tuple_cost = 0.001	indica el coste de procesar una entrada en una página de índice (0.001)	SET cpu_index_tuple_cost TO valor_decimal
cpu_operator_cost = 0.0025	indica el coste de procesar un operador (0.0025)	SET cpu_operator_cost TO valor_decimal
GEQO (Genetic Query Optimizer)		
geqo = on	si PostgreSQL usa el GEQO para eliminar aquellos planes de ejecución que posiblemente sean caros	ALTER DATABASE SET geqo TO [on | off]
geqo_threshold = 12	indica el nº de items en el FROM a partir del cual se debe usar GEQO	ALTER DATABASE SET geqo_thersold TO número
geqo_effort = 5	controla el compromiso entre el tiempo de planificación y la eficiencia del plan de ejecución (entero entre 1 y 10)	ALTER DATABASE SET geqo_effort TO número
geqo_pool_size = 0	indica el nº de individuos de una población (al menos 2, valor típico entre 100 y 1000). Si es 0, el valor por defecto adecuado se elige en función de geqo_effort	ALTER DATABASE SET geqo_pool_size TO número
geqo_generations = 0	indica el valor para las generaciones (iteraciones del algoritmo). Es un valor entero, si es 0 se calcula con la fórmula: $geqo\_effort*Log_2(geqo\_pool\_size)$	
geqo_selection_bias = 2.0	indica la presión selectiva dentro de la población (entre 1.5 y 2.0)	SET geqo_selection_bias TO valor_decimal
Otras opciones		
default_statistics_target = 10	establece el objetivo estadístico por defecto para los columnas de las tablas que no lo tienen definido explícitamente con ALTER TABLE SET STATISTICS).	
constraint_exclusion = off	establece si el optmizador debe utilizar las restricciones para realizar la optimización	

parámetro	uso	modificación
from_collapse_limit = 8	establece si el optimizador debe fusionar subconsultas en las consultas si los items en el FROM está por debajo del valor dado	
join_collapse_limit = 8		

8.7 Errores / Seguimiento

parámetro	uso	modificación
Lugar de seguimiento		
log_destination = 'stderr'	existen varios métodos para emitir los mensajes del servidor (stderr, syslog y eventlog)	fichero de configuración
redirect_stderr = on	permite enviar los errores enviados a stderr a los ficheros de seguimiento	puesta en marcha del servidor
log_directory = 'pg_log'	si el parámetro anterior está habilitado determina el directorio donde se crean los ficheros de seguimiento	puesta en marcha del servidor
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'	si redirect_stderr = on, nombre de los ficheros de seguimiento	
log_truncate_on_rotation = off	establece la rotación de ficheros, aunque se pueden usar alternativas	
log_rotation_age = 1440	si redirect_stderr = on establece la duración máxima de cada fichero de seguimiento, con 0 deshabilita esta opción	
log_rotation_size = 10240	tamaño de los ficheros rotados	
syslog_facility = 'LOCAL0'	si el seguimiento lo hace syslog, aquí se determina qué utilidad se usa	
syslog_ident = 'postgres'	si se usa syslog este es el nombre del programa utilizado para identificar los mensajes de PostgreSQL.	

Cuándo efectuar el seguimiento

en algunas opciones, se establecen níveles de mensaje, que en el propio fichero de configuración están los posibles valores, elegir un determinado nivel incluye todos los niveles que se encuentran por debajo. Los niveles que existen son:

- `DEBUG[1-5]: información para desarrolladores`.
- `INFO`: información implícita sobre operaciones del usuario, por ej. durante `VACUUM VERBOSE`.
- `NOTICE`: información que puede ayudar a los usuarios, por ej., en la creación de índices como parte de la clave primaria.
- `WARNING: avisos al usuario, caso de un commit fuera de un bloque de transacción`.
- `ERROR`: informa de un error que ha causado que aborte un comando.
- `LOG`: información interesante para los administradores, por ej, la actividad de los puntos de verificación (checkpoints).
- `FATAL`: errores que han producido que aborte la sesión.
- `PANIC`: errores que han producido que aborten todas las sesiones.

client_min_messages = notice	establece el nivel de los mensajes que serán enviados a los clientes	puesta en marcha del servidor
log_min_messages = notice	controla el nivel de los mensajes que son escritos en el fichero de seguimiento	puesta en marcha del servidor
log_error_verbosity = default	controla el detalle de la información que se escribe en el fichero de seguimiento (terse, default, verbose), cada nivel añade más campos	

parámetro	uso	modificación
log_min_error_statement = panic	controla si la instrucción SQL que ha provocado el error debe ser recordada o no el el fichero de seguimiento. En caso de querer hacer un seguimiento, conviene cambiarla a ERROR.	puesta en marcha del servidor
log_min_duration_statement = -1	registra las instrucciones y su duración si su ejecución tarda más que el indicado aquí. Con 0 registra todas y con -1 ninguna	puesta en marcha del servidor
silent_mode = off	la salida estándar y los errores se envían a /dev/null	
Qué se registra		
debug_print_parse = off	configuran el servidor para que registre de cada consulta información de su ejecución	puesta en marcha del servidor y con SET
debug_print_rewritten = off		puesta en marcha del servidor
debug_print_plan = off		puesta en marcha del servidor
debug_pretty_print = off		puesta en marcha del servidor
log_connections = off	información detallada de cada conexión	puesta en marcha del servidor
log_disconnections = off	información detallada de cada desconexión	puesta en marcha del servidor y con SET
log_duration = off	registra la duración de cada instrucción que va a ser registrada	puesta en marcha del servidor y con SET
log_line_prefix =	escribe una cadena antes de cada línea de informe.	puesta en marcha del servidor y con SET
log_statement = 'none'	controla qué instrucciones son registradas (none, ddl, mod y all) y registra informaci	puesta en marcha del servidor y con SET
log_hostname = off	con esta opción, aparte de la IP, se registra también el nombre del host desde donde se establece la conexión	puesta en marcha del servidor

8.8 Estadísticas

parámetro	uso	modificación
Monitorización		
log_parser_stats = off log_planner_stats = off log_executor_stats = off log_statement_stats = off	registran estadísticas de seguimiento de cada módulo para cada consulta. Sólo se usa para depuración	puesta en marcha del servidor y con SET
Recolección de estadísticas de consultas e índices		
stats_start_collector = on	si se lanza el subproceso de recolección de estadísticas	puesta en marcha del servidor y con SET
stats_command_string = off	habilita la recolección de estadísticas de cada instrucción ejecutada	puesta en marcha del servidor y con SET
stats_block_level = off	habilita la recolección de estadísticas de bloque	puesta en marcha del servidor y con SET
stats_row_level = on	habilita la recolección de estadísticas de fila	puesta en marcha del servidor y con SET
stats_reset_on_server_start = off	la información estadística se reinicia en la puesta en marcha del servidor	puesta en marcha del servidor

8.9 Vacuum (purga)

parámetro	uso	modificación
autovacuum = on	si el servidor debe lanzar el proceso de auto-purga, *autovacuum*	puesta en marcha del servidor

parámetro	uso	modificación
autovacuum_naptime = 60	periodo de parada en segundos entre dos puestas en marcha del autovacuum	puesta en marcha del servidor
autovacuum_vacuum_threshold = 1000	nº mínimo de filas modificadas o borradas para lanzar el autovacuum. Este valor se puede sobreescribir para tablas individuales incluyendo información en *pg_autovacuum.*	puesta en marcha del servidor
autovacuum_analyze_threshold = 500	especifica el nº mínimo de filas modificadas o borradas para lanzar un *analyze* sobre una tabla. Este valor se puede sobreescribir para tablas individuales incluyendo información en *pg_autovacuum.*	
autovacuum_vacuum_scale_factor = 0.4	especifica la fracción del tanmaño de la tabla que se debe añadir a *autovacuum_vacuum_thershold* para decidir si se lanza la purga.	
autovacuum_analyze_scale_factor = 0.2	especifica la fracción del tanmaño de la tabla que se debe añadir a *autovacuum_analyze_thershold* para decidir si se lanza el *analyze.*	
autovacuum_vacuum_cost_delay = -1	especifica el coste de retraso que se debe utilizar en las operaciones autovacuum. Si es -1 se usa *vacuum_cost_delay*. Este valor se puede sobreescribir para tablas individuales incluyendo información en *pg_autovacuum.*	
autovacuum_analyze_cost_delay = -1	especifica el coste de retraso que se debe utilizar en las operaciones analyze. Si es -1 se usa *analyze_cost_delay*. Este valor se puede sobreescribir para tablas individuales incluyendo información en *pg_autovacuum.*	

8.10 Conexión cliente

parámetro	uso	modificación
Comportamiento		
search_path = '$user,public'	órden de búsqueda de esquemas de la base de datos	fichero postgresql.conf y SET
default_tablespace = ''	tablespace donde se crearan los objetos si no se indica otro	puesta en marcha del servidor y SET
check_function_bodies = on	habilita la validación de los cuerpos de las funciones que se crean	puesta en marcha del servidor y SET
default_transaction_isolation = 'read committed'	establece el nivel de aislamiento por defecto	SET
default_transaction_read_only = off	establece cómo son las nuevas transacciones, 'read_only' o 'read/write'. Las transacciones read_only solo pueden modificar tablas temporales.	SET
statement_timeout = 0	se abortará cualquier instrucción cuya ejecución supere este límite. 0 deshabilita esta opción.	SET

parámetro	uso	modificación
Localización y formato		
datestyle = 'iso, mdy' timezone = unknown australian_timezones = off extra_float_digits = 0 client_encoding = sql_ascii lc_messages = 'Es_ES@euro' lc_monetary = 'Es_ES@euro' lc_numeric = 'Es_ES@euro' lc_time = 'Es_ES@euro'	comportamiento del servidor asociado a valores geográficos y de tiempo. Si no se configuran , se usan los valores del entorno. cambiar datestyle = 'postgres, european'	SET

8.11 Gestión de Bloqueos

parámetro	uso	modificación
deadlock_timeout = 1000	tiempo en milisegundos que el servidor espera cuando se genera un bloqueo para comprobar la condición de abrazo mortal.	puesta en marcha del servidor
max_locks_per_transaction = 64	la tabla de bloqueos compartida se crea para alojar un nº de bloqueos: *max_locks_per_transactions *(max_conections + max_prepared_transaction)*	

8.12 Opciones predefinidas

Estas opciones sólo se pueden cambiar en el momento de la compilación, cuando ejecutamos *configure.*

parámetro	uso
block_size = 8192	tamaño de bloque de base de datos. El valor es el que se indica en l momento de la instalación.
integer_datetime = off	fechas e instantes con soporte de 64 bits o no. El valor es el que se indica en l momento de la instalación.
lc_collate lc_ctype	forma de hacer la ordenación y las clasificaciones de texto. El valor es el que indica el cluster de BD
max_function_args = 100	nº máximo de argumentos en las funciones. El valor es el que se indica en l momento de la instalación.
max_identifier_length = 63	longitud máxima de las cadenas de los identificadores. El valor es el que se indica en l momento de la instalación.
max_ident_keys = 32	nº máximo de columnas en la clave de indexación. El valor es el que se indica en l momento de la instalación.
server_encoding	juego de caracteres, por defecto el que indique la base de datos.
server_version	nº de versión del servidor.

Capítulo 9. Internacionalización y localización

Conceptos:

- **Internacionalización:** proceso de desarrollo de software de forma que pueda ser usado en distintas ubicaciones.
- **Localización:** proceso de adaptación de una aplicación para que sea usada en una ubicación específica.

Ventajas:

- mensajes en varios idiomas
- mensajes en varios juegos de caracteres
- visualización de datos de usuario en diversos juegos de caracteres
- diversos tipos de ordenaciones
- clasificación de caracteres: mayúsculas, puntuación, acentos, etc.

El soporte local consiste en un conjunto nominado de propiedades que definen las convenciones culturales específicas de un lugar. Se hace uso de las facilidades de soporte local que proporciona el sistema operativo. Se suele mostrar de la siguiente forma:

<idioma>_<ubicación>.<juego_de_caracteres>@<modificadores>

Para ver cómo está configurado en el SO ejecutar:

```
$ locale
```

que nos mostrará la configuración activa, podemos cambiar la configuración entre cualquiera de las que estén instaladas en el sistema operativo, para ver cuáles están disponibles:

```
$ locale –a
```

ejemplos de valores que salen:

```
Linux:      es_ES@euro
            es_ES.UTF8
            es_ES.UTF8@euro

Windows:    es_ES@CP1252
```

PostgreSQL soporta estas opciones de localización y de juegos de caracteres (admite hasta conjuntos de caracteres multibyte), pero se debe haber habilitado la opcion de soporte local cuando se realizó la instalación, concretamente los parámetros *"enable-nls" y "enable-locale"* cuando ejecutamos "*configure".*

Para ver cómo lo hicimos, ejecutar, con el usuario postgres o con el que se instaló:

```
$ pg_config --configure
```

¡¡¡ Si no esta el soporte, toca reinstalar !!!

Respecto a la localización, las variables de entorno del SO que influyen son:

- LC_MESSAGES: formato y lenguaje de los mensajes.
- LC_MONETARY: formato de los valores monetarios.
- LC_NUMERIC: formato de los valores numéricos.
- LC_TIME: formato de los valores de fecha y tiempo.
- LC_CTYPE: clasificación de caracteres (mayúsculas, puntuación, etc.)
- LC_COLLATE: ordenación de los valores de tiras de caracteres.
- LC_ALL: todos los anteriores.

En la localización influye:

- la localización del cliente: viene dada por la configuración del entorno o por forma de llamar a un programa.
- la localización del servidor: viene dado por la configuración del entorno y por el fichero de configuración.
- la localización del cluster de base de datos.: se asigna al ejecutar *initdb*, y una vez creado, los parámetros LC_COLLATE y LC_CTYPE no se puede cambiar.

En cuanto a los juegos de caracteres:

- soporta distintos tipos: byte simple o multibyte.
- el cluster puede estar en un conjunto de caracteres y la base de datos en otro:

```
$ initdb –E unicode
$ createdb –E latin1 base1
SQL# create database base2 with encoding ‘win’;
```

- se produce una conversión automática entre servidor / cliente (en la tabla *pg_conversion* están las conversiones por defecto.
- se puede cambiar el juego de caracteres del cliente de varios modos:

```
psql:

\encoding -<valor>

SQL:

set client_encoding to ‘valor’;
```

Capítulo 10. Estructura de PostgreSQL II

10.1 Procesamiento de instrucciones

Se produce un diálogo básico entre el proceso front-end (aplicación) y el proceso back-end (postgres), que consta básicamente de los siguientes pasos:

1. La aplicación **envía** una instrucción al proceso postgres.
2. El servidor realiza en **análisis** y el proceso postgres responde OK o devuelve error.
3. La aplicación solicita la **ejecución** y el servidor la realiza, devolviendo Ok o error.
4. Por último la aplicación solicita la **recuperación** de los datos y el servidor responde con los resultados de la instrucción.

Para el procesamiento de instrucciones PostgreSQL usa la **caché de la base de datos (*shared buffers)***, colección de buffers de 8Kb en la memoria compartida y que sirven como caché de las páginas de los ficheros y sentencias que se han usado recientemente. Esta memoria se reserva cuando se pone en marcha el servidor.

Los bloques de buffers modificados se llaman ***"sucios"*** y se colocan en una ***lista sucia***. La lista sucia controla todas las modificaciones de datos que se hacen en la caché y que no se han vaciado al disco. La lista sucia se vacía, por ejemplo, cuando se produce un punto de control (checkpoint) o simplemente se llena la lista.

Si la instrucción es una consulta, el servidor siempre lee primero si tiene estos bloques de datos en la caché, en caso contrario, los lee de disco y los copia a la cache, realizando esta operación tantas veces como sea necesario para ejecutar una instrucción.

Si la instrucción es una actualización de datos entran en juego también los **ficheros de diario** (**WAL**) y los **buffers de diario (buffers WAL)**. Por ejemplo, al ejecutar un Update de una fila:

1. mirar si los datos están en los buffers compartidos y recuperar el valor antiguo
2. si no están recuperarlos de los ficheros de datos y ponerlos en los shared buffers
3. se escribe el valor nuevo en el buffer WAL
4. se realiza la modiciación y se escribe en los shared buffers

¡todavía no se han escrito los datos en disco!

10.2 Gestión de transacciones

PostgreSQL cumple las reglas **ACID**:

- Atomicidad (atomicity)
- Consistencia (consistency)
- Aislamiento (isolation)
- Persistencia (durability)

10.2.1 Atomicidad

La palabra se deriva de *átomo*, que significa indivisible; como indivisibles son cada una de las operaciones contenidas dentro de una transacción. Una transacción es un bloque de operaciones o instrucciones dirigido a modificar una base de datos en una determinada dirección.

En realidad, lo importante aquí, es que se cumpla la regla de *todo o nada*. Si todas las operaciones se realizan con éxito, los resultados producidos por la transacción se guardan en la base de datos. Si alguna operación falla, todas las operaciones son deshechas (rollback), incluso aunque alguna operación haya sido realizada con éxito.

Solución

En PostgreSQL podemos hacer que un grupo de sentencias SQL se ejecuten dentro de una transacción, encerrándolas entre las sentencias ***BEGIN*** y ***COMMIT***. De esta forma aseguramos que se ejecutan todas o ninguna. Si llegados a algún punto dentro de la transacción necesitamos deshacerla completamente, utilizamos ***ROLLBACK***, en lugar de COMMIT, y todos los cambios son deshechos.

Durante las transacciones, existen restricciones ***diferidas*** (ser verifican al final de la transacción) o ***inmediatas*** (se verifican en cada operación), en este caso, si alguna condición no se cumple, la transacción queda en estado latente, algunas consultas no se pueden realizar y las actualizaciones no tendrán efecto, con lo que solo se puede resolver deshaciendo la transacción.

Por otro lado, PostgreSQL trata cualquier sentencia aislada como si ésta se ejecutara dentro de una pequeña transacción; aunque no usemos BEGIN, cada sentencia incorpora implícitamente un BEGIN y, si culmina con éxito, un COMMIT, a este modo se le conoce como ***"modo autoconfirmación"***.

Además, existen los ***savepoints*** que permiten anular partes de una transacción, por ejemplo:

```
BEGIN;
… operaciones1
SAVEPOINT paso2;
… operaciones2
… si error => ROLLBACK paso2;
… operaciones3
COMMIT;
```

de modo que si se llega al commit, quedan confirmadas operaciones1 y operaciones3, y operaciones2 quedará confirmada si no ha habido error.

10.2.2 Consistencia.

Una base de datos es una herramienta para modelar y registrar una realidad cambiante. Sin embargo, debe hacerlo *consistentemente* de acuerdo con unas reglas determinadas o restricciones de integridad definidas. La realidad modelada así, puede ir cambiando al pasar de un estado en el que cumple las reglas, a otro en el que también las cumple.

Si al ejecutar una transacción, por el motivo que sea, se incumpliera alguna de estas reglas, la transacción no se llegaría a completar. Esto garantiza que una base de datos siempre permanezca en un estado válido, es decir, *consistente* con las reglas.

Solución:

A partir de la versión 7.0 se añadió gestión de integridad en PostgreSQL, las restricciones soportadas son:

- *not null*
- *check*
- *unique*
- *primary key*
- *foreign key*, con dos tipos "*match full*" (completa) y "*match partial*" (si en alguna columna no hay valores no se comprueba la integridad, aunque esta opción no está implementada)

Solo las instrucciones de clave ajena son diferibles, el resto son inmediatas. Las sentencias de creación de claves ajenas admiten estos parámetros:

[DEFERRABLE] [INITIALLY {INMEDIATE | DEFERRED }]

- Inmediate: se comprueba la consistencia después de cualquier sentencia SQL.
- Deferred: se comprueba la consistencia cuando termina la transacción, al final.
- Deferrable: indica que el punto de comprobación se puede cambiar dinámicamente.

10.2.3 Aislamiento

Los resultados de una transacción que está siendo ejecutada, son invisibles a cualquier otra transacción hasta que la primera no haya sido completada con éxito. La ejecución no perturba a otras transacciones que se ejecuten concurrentemente.

Solución:

Frente a otro modelos tradicionales que controlan el acceso concurrente a los datos a través de bloqueos, PostgreSQL utiliza un modelo denominado ***Multiversion Concurrency Control (MVCC)***, según el cual, al consultar la BD, cada transacción ve una instantánea de la BD tal como era hace un cierto tiempo (una versión de la BD) y no el estado actual de la BD. Este mecanismo evita que una transacción pueda ver datos inconsistentes modificados por otra. Aislando las transacciones que operan concurrentemente en distintas sesiones, un lector no necesita esperar a un escritor; y un escritor no necesita esperar a un lector.

10.2.4 Persistencia o Durabilidad

La durabilidad garantiza que los efectos resultantes de una transacción, una vez ha sido completada con éxito, permanecen en la base de datos para siempre, incluso cuando se puedan producir posteriormente fallos de cualquier clase.

Por ejemplo, si durante una transacción se produce un apagado del servodor, una vez reiniciado el servidor, un sistema que lleve un registro de transacciones realizadas, advertiría rápidamente que existe una sin completar, finalizándola correctamente.

Solución:

PostgreSQL utiliza una técnica estándar denominada **WAL (Write-ahead logging, o escritura anticipada de registros)** para controlar tanto la consistencia como la durabilidad de las transacciones.

Brevemente explicada, consiste en que los cambios en los ficheros de datos (tablas e índices) sólo se materializan cuando existe previamente en el disco un registro en el que están anotados dichos cambios. Siguiendo este procedimiento, no es necesario enviar páginas al disco cada vez que una transacción se completa. Esta técnica no sólo mejora el rendimiento del servidor, sino que ante un fallo de la máquina, será posible recuperar la BD a partir de ese registro: cualquier cambio no aplicado a las páginas de datos en el disco será nuevamente hecho desde el log (*roll-forward recovery, o **REDO***) mientras que los posibles cambios realizados en páginas de disco por transacciones incompletas, podrán ser deshechos manteniendo la integridad y la consistencia de los datos (*roll-backward recovery, o **UNDO***).

Estos ficheros de diario están en $PGDATA/pg_xlog, son ficheros de 16Mb, divididos en páginas de 8Kb, el sistema crea uno inicialmente y va creando más según las necesidades, rotando de manera cíclica según marquen los parámetros de sistema que gestionan este funcionamiento (checkpoint_segments y archive_command).

Siguiendo con la instrucción vista en el punto 10.1 (update de una fila), al ejecutar un COMMIT se produce:

1. Se escriben los datos del buffer WAL en los ficheros de diario WAL
2. Se pueden refrescar las copias multiversión en los shared buffers
3. Se retorna el control a la aplicación

Entonces, los datos de los buffers no se han escrito en disco todavía, pero los del diario si, así los datos de la transacción nunca se pierden.

10.3 Más sobre MVCC

Multi-Version Concurrency Control (MVCC) es una técnica avanzada para mejorar las prestaciones de una base de datos en un entorno multiusuario.

La principal diferencia entre multiversión y el modelo de bloqueo es que en los bloqueos MVCC derivados de una consulta (lectura) de datos no entran en conflicto con los bloqueos derivados de la escritura de datos y de este modo la lectura nunca bloquea la escritura y la escritura nunca bloquea la lectura.

Aislamiento transaccional

El estándar ANSI/ISO SQL define cuatro niveles de aislamiento transaccional en función de tres hechos que deben ser tenidos en cuenta entre transacciones concurrentes. Estos **hechos no deseados** son:

- **lecturas "sucias":** una transacción lee datos escritos por una transacción no esperada, no confirmada.
- **lecturas no repetibles**: una transacción vuelve a leer datos que previamente había leído y encuentra que han sido modificados por una transacción confirmada. Indica que, varias sentencias SELECT, dentro de la misma transacción devuelven resultados distintos.
- **lectura "fantasma"**: una transacción vuelve a ejecutar una consulta, devolviendo un conjuto de filas que satisfacen una condición de búsqueda y encuentra que otras filas que satisfacen la condición han sido insertadas por otra transacción confirmada. Dentro de una transacción, buscamos un registro y a veces lo encontramos y a veces no.

Los **cuatro niveles de aislamiento** y sus correspondientes acciones se describen más abajo.

	Lectura "sucia"	**Lectura no repetible**	**Lectura "fantasma"**	**Bloqueos**
Lectura no confirmada	Posible	Posible	Posible	Nunca
Lectura confirmada	No posible	Posible	Posible	Si, si hay actualizaciones
Lectura repetible	No posible	No posible	Posible	Si, si hay SELECTs
Serializable	No posible	No posible	No posible	Siempre, es fácil provocar un deadlock

PostgreSQL ofrece

- **lectura confirmada (read_commited)**:

  Nivel por defecto (ver parámetro "default_transaction_isolation"), en una transacción T sólo pueden leer datos cambiados por otras transacciones que estén confirmados en el momento de la lectura. Cuando una transacción se ejecuta en este nivel, la consulta sólo ve datos confirmados antes de que la consulta comenzara y nunca ve ni datos "sucios" ni los cambios en transacciones concurrentes confirmados durante la ejecución de la consulta.

  Si una fila devuelta por una consulta mientras se ejecuta una sentencia **UPDATE** (o **DELETE**, o **SELECT FOR UPDATE**) está siendo actualizada por una transacción concurrente no confirmada, entonces la segunda transacción que intente actualizar esta fila esperará a que la otra transacción se confirme o pare. En caso de que pare, la transacción que espera puede proceder a cambiar la fila. En caso de que se confirme (y si la fila todavía existe, por ejemplo, no ha sido borrada por la otra transacción), la consulta será reejecutada para esta fila y se comprobará que la nueva fila satisface la condición de búsqueda de la consulta. Si la nueva versión de la fila satisface la condición, será actualizada (o borrada, o marcada para ser actualizada).

- **serializable (serial):**

  En una transacción T, sólo se pueden leer datos cambiados por otras transacciones que estén confirmados que estén confirmados previamente al inicio de T. En las actualizaciones se usan bloqueos para serializar la aplicación, esto produce mayor coste de recursos si hay muchas sesiones.

  La *serialización* proporciona el nivel más alto de aislamiento transaccional. Cuando una transacción está en el nivel serializable, la consulta sólo ve los datos confirmados antes de que la transacción comience y nunca ve ni datos sucios ni los cambios de transacciones concurrentes confirmados durante la ejecución de la transacción. Por lo tanto, este nivel emula la ejecución de transacciones en serie, como si las transacciones fueran ejecutadas una detrás de otra, en serie, en lugar de concurrentemente.

  Si una fila devuelta por una consulta durante la ejecución de una sentencia **UPDATE** (o **DELETE**, o **SELECT FOR UPDATE**) está siendo actualizada por una transacción concurrente no confirmada, la segunda transacción que trata de actualizar esta fila esperará a que la otra transacción se confirme o pare. En caso de que pare, la transacción que espera puede proceder a cambiar la fila. En el caso de una transacción concurrente se confirme, una transacción serializable será parada con el mensaje:

```
ERROR:  Can't serialize access due to concurrent update
```

  porque una transacción serializable no puede modificar filas cambiadas por otras transacciones después de que la transacción serializable haya empezado.

10.4 Bloqueos y tablas

Postgres ofrece varios modos de bloqueo para controlar el acceso concurrente a los datos en tablas. Algunos de estos modos de bloqueo los adquiere Postgres automáticamente antes de la ejecución de una sentencia, mientras que otros son proporcionados para ser usados por las aplicaciones. Todos los modos de bloqueo (excepto para AccessShareLock) adquiridos en un transacción se mantienen hasta la duración de la transacción.

Además de bloqueos, también se usa compartición en exclusiva para controlar acessos de lectura/escritura a las páginas de tablas en un buffer compartido. Este método se pone en marcha inmediatamente después de que una fila es traída o actualizada.

Bloqueos a nivel de tabla

AccessShareLock

Un modo de bloqueo adquirido automáticamente sobre tablas que están siendo consultadas. Postgres libera estos bloqueos después de que se haya ejecutado una sentencia.

Conflictos con AccessExclusiveLock.

RowShareLock

Adquirido por **SELECT FOR UPDATE** y **LOCK TABLE** para declaraciones `IN ROW SHARE MODE`.

Entra en conflictos con los modos ExclusiveLock y AccessExclusiveLock.

RowExclusiveLock

Lo adquieren **UPDATE**, **DELETE**, **INSERT** y **LOCK TABLE** para declaraciones `IN ROW EXCLUSIVE MODE`.

Choca con los modos ShareLock, ShareRowExclusiveLock, ExclusiveLock y AccessExclusiveLock.

ShareLock

Lo adquieren **CREATE INDEX** y **LOCK TABLE** para declaraciones `IN SHARE MODE`.

Está en conflicto con los modos RowExclusiveLock, ShareRowExclusiveLock, ExclusiveLock y AccessExclusiveLock.

ShareRowExclusiveLock

Lo toma **LOCK TABLE** para declaraciones `IN SHARE ROW EXCLUSIVE MODE`.

Está en conflicto con los modos RowExclusiveLock, ShareLock, ShareRowExclusiveLock, ExclusiveLock y AccessExclusiveLock.

ExclusiveLock

Lo toma **LOCK TABLE** para declaraciones `IN EXCLUSIVE MODE`.

Entra en conflicto con los modos RowShareLock, RowExclusiveLock, ShareLock, ShareRowExclusiveLock, ExclusiveLock y AccessExclusiveLock.

AccessExclusiveLock

Lo toman ALTER TABLE, DROP TABLE, VACUUM y LOCK TABLE.

Choca con RowShareLock, RowExclusiveLock, ShareLock, ShareRowExclusiveLock, ExclusiveLock y AccessExclusiveLock.

Sólo AccessExclusiveLock bloquea la sentencia **SELECT** (sin `FOR UPDATE`)

Bloqueos a nivel de fila

Este tipo de bloqueos se producen cuando campos internos de una fila son actualizados (o borrados o marcados para ser actualizados). Postgres no retiene en memoria ninguna información sobre filas modificadas y de este modo no tiene límites para el número de filas bloqueadas sin incremento de bloqueo.

Sin embargo, hay que tener en cuenta que **SELECT FOR UPDATE** modificará las filas seleccionadas marcándolas, de tal modo que se escribirán en el disco.

Los bloqueos a nivel de fila no afecta a los datos consultados. Estos son usados para bloquear escrituras *a la misma fila* únicamente.

10.5 Bloqueo e índices

Aunque Postgres proporciona desbloqueo para lectura/escritura de datos en tablas, no ocurre así para cada método de acceso al índice implementado en en Postgres.

Los diferentes tipos de índices son manejados de la siguiente manera:

Indices GiST y R-Tree

Nivel de bloqueo de índice del tipo Compartición/exclusividad para acceso lectura/escritura. El bloqueo tiene lugar después de que la sentencia se haya ejecutado.

Indices hash

Se usa el bloqueo a nivel de página para acceso lectura/escritura. El bloqueo tiene lugar después de que la página haya sido procesada.

Los bloqueos a nivel de página producen mejor concurrencia que los bloqueos a nivel de índice pero pueden provocar "puntos muertos".

Btree

Se usan bloqueos a nivel de página de compartición/exclusividad en los accesos de lectura/escritura. Los bloqueos se llevan a cabo inmediatamente después de que el fila índice sea insertado o buscado.

Los índices Btree proporciona la más alta concurrencia sin provocar "estados muertos".

10.6 Chequeos de consistencia de datos en el nivel de aplicación

Ya que las lecturas en Postgres no bloquean los datos, sin tener en cuenta el nivel de aislamiento de la transacción, los datos leídos por una transacción pueden ser sobreescritos por otra. En otras palabras, si una fila es devuelta por **SELECT** esto no significa que esta fila realmente exista en el momento en que se devolvió (un tiempo después de que la sentencia o la transacción comenzaran, por ejemplo) ni que la fila esté protegida de borrados o actualizaciones por la transacción concurrente antes de que ésta se lleve a cabo o se pare.

Para asegurarse de la existencia de una fila y protegerla contra actualizaciones concurrentes, debería usar **SELECT FOR UPDATE** o una sentencia de tipo **LOCK TABLE** más apropiada. Esto debe tenerse en cuenta cuando desde otros entornos se estén portando aplicaciones hacia Postgres utilizando el modo serializable

.

Capítulo 11. Seguridad en PostgreSQL

La seguridad en PostgreSQL se materializa en tres aspectos:

- Seguridad en la manipulación de los ficheros de PostgreSQL.
- Seguridad en los accesos de los clientes.
- Definición de los privilegios para acceder a los objetos de la base de datos a los usuarios PostgreSQL.

11.1 Seguridad en la manipulación de los ficheros

La información más crítica está en $PGDATA.

- todos los ficheros deben pertenecer al usuario *postgres*.
- el usuario *postgres* debe ser el único que pueda:
  - leer, escribir y ejecutar sobre los directorios en los ficheros
  - leer y escribir en los ficheros
- respecto a las conexiones locales vía sockets:
  - todos los usuarios usan la misma
  - se puede restringir el uso para algunos usuarios del SO
  - ver el directorio /tmp, donde existe un fichero *.s.PGSQL.5432* que se crea cuando arranca el postmaster, los permisos de este fichero son 777, cualquier usuario del SO se puede conectar. Además hay un *.s.PGSQL.5432.lock* que sólo puede leer *postgres.*
  - los permisos de este socket se pueden configurar en *postgresql.conf* con los parámetros *unix_socket_directory, unix_socket_group y unix_socket_permission.*

11.2 Seguridad en el acceso de los clientes

Es importante poder definir desde qué equipos se pueden conectar a nuestra base de datos, así como poder definir qué usuarios y a qué bases de datos se pueden conectar.

La configuración de este nivel de seguridad se realiza en los ficheros ***pg_hba.conf*** *(hba = host based authentication) y* ***pg_ident.conf.***

Se trata de editar una serie de reglas que se irán procesando de arriba abajo, cuando se encuentre una regla que cumpla la conexión, se ejecutará lo que ponga en el método.

Hay cuatro formas generales de definir un acceso autorizado:

TIPO	BASE DATOS	USUARIOS	DIRECCIÓN	MÉTODO	
LOCAL	base_datos	usuario		método-autenticación	[opción]
HOST	base_datos	usuario	direcciónCIDR	método autenticación	[opción]
HOSTSSL	base_datos	usuario	direcciónCIDR	método autenticación	[opción]
HOSTNOSSL	base_datos	usuario	direcciónCIDR	método autenticación	[opción]

11.2.1 Conexión local: usando los sockets del dominio Unix

```
TIPO        BASE DATOS  USUARIOS    DIRECCIÓN   MÉTODO
LOCAL       base_datos  usuario                 método-autenticación    [opción]
```

base_datos:

- **ALL**: se permite la conexión a cualquier base de datos
- **SAMEUSER**: solo a bases de datos que su nombre sea el mismo que el usuario que se conecta
- **SAMEROLE**: solo a bases de datos que su nombre sea el mismo que el role que se conecta
- **nombd1, nombd2,…**: se permite la conexión a cualquiera de las bases de datos de la lista
- **@fichero**: se permite la conexión a las bases de datos incluidas en el fichero, que debe estar en el mismo directorio que pg_hba.conf

usuario:

- **ALL**: se permite la conexión de cualquier role
- **role1, [+]role2,…:** se permite la conexión de los roles de la lista y además se permite la conexión de cualquier role que sea miembre de role2
- **@fichero**: se permite la conexión de los roles incluidos en el fichero, que debe estar en el mismo directorio que pg_hba.conf

método-autenticación

- **TRUST:** conexión aceptada sin condiciones
- **REJECT**: conexión rechazada sin condiciones
- **PASSWORD**: se solicita palabra de paso sin encriptar, las palabras de paso se almacenan en la tabla *pg_authid* y pueden estar cifradas o no según como se creara el role.
- **CRYPT**: palabra de paso encriptada (versiones previas a la 7.2)
- **MD5**: palabra de paso con el método de encriptación md5, y se almacena también con este método. Es el método recomendado por PostgreSQL. Se obtiene un cifrado a partir de la ID de usuario y la palabra de paso, el cliente solicita una semilla al servidor y así se obtiene un segundo cifrado que es envíado al servidor, en el servidor se utiliza la palabra de paso almacenada, la ID del usuario (la obtiene de la conexión) y la semilla para obtener un cifrado similar y los compara.
- **KRB5**: se usa Kerberos v5 para autenticar el cliente, se ha de habilitar en la instalación del servidor.

- **IDENT correspondencia**: a partir del usuario de la conexión cliente (se fía de la autenticación del cliente) y de la correspondencia indicada en la opción, se obtiene un role de PostgreSQL para realizar la conexión. Las correspondencias se obtienen del fichero *pg_ident.conf.* La correspondencia puede ser:
    - **SAMEUSER:** el usuario del sistema operativo es el mismo y solo el mismo que se conecta a la BD.
    - **cambio-usuario:** el sistema mira el fichero *pg_ident.conf,* y busca una fila donde esté la correspondencia llamada 'cambio-usuario' y se corresponda con el usuario conectado al SO, haciendo la conexión a la BD con el usuario con el usuario de la columna *usuario-pg.*
- **PAM servicio-pam**: autenticación usando Pluggable Authentication Method proporcionado por el servicio PAM indicado en opción. Este método es proporcionado por el SO. Se debe compilar el servidor con esta opción. El PAM por defecto es *postgresql.* Se deben configurar los métodos PAM del SO, generalmente basta con incluir la definición del un nuevo método en el directorio */etc/pam.d.*

Ejemplo de uso de PAM:

1. copiar el método */etc/pam.d/passwd* con nombre *postgresql*
2. dar permisos a *postgres* para leer el fichero de contraseñas
    - crear un grupo del SO llamado vershadow
    - incorporar al usuario *postgres* al grupo *vershadow*
    - cambiar */etc/shadow* para que su grupo sea *vershadow*
    - cambiar los permisos de acceso de */etc/shadow* para que puedan acceder a él en lectura miembros del grupo (740)
    - añadir al *pg_hba.conf* algún acceso con PAM
    - tirar y levantar el servidor PostgreSQL.

11.2.2 Conexión remota sin encriptar usando TCP/IP (SSL y no SSL)

```
TIPO  BASE DATOS  USUARIOS    DIRECCIÓN         MÉTODO
HOST  base_datos  usuario     direcciónCIDR     método autenticación    [opción]
```

- permite conexiones SSL y no SSL.
- la palabra de paso se transmite en texto plano si es no SSL
- direccionCIDR ó direcccionIP máscara IP: especifica un rango de direcciones IP
    - 192.168.200.0/24 ó 192.168.200.0 255.255.255.0: se pueden conectar todas las IPs de la red 192.168.200
    - 192.168.0.0/16 ó 192.168.0.0 255.255.0.0: todos las IPs de la red 192.168
    - 192.168.200.85/32: solo esa IP
    - 0..0.0.0/0 ó 0.0.0.0 0.0.0.0.: cualquier IP

11.2.3 Conexión remota encriptada SSL usando TCP/IP (solo SSL)

```
TIPO        BASE DATOS  USUARIOS    DIRECCIÓN         MÉTODO
HOSTSSL     base_datos  usuario     direcciónCIDR     método autenti.   [opción]
```

- la transmisión cliente-servidor es encriptada
- los registros HOST se utilizarán para conexiones remotas no-SSL y SSL
- los registros HOSTSSL sólo se utilizarán para conexiones remotas SSL
- requiere instalar PostgreSQL con la opción SSL, crear los ficheros de firmas y claves y ubicarlos en el lugar correspondiente

Para configurar el servidor para estas opciones:

- SSL tiene que estar instalado en el servidor y en los clientes que se conecten)
- PostgreSQL debe estar compilado con la opción SSL

```
$ ./configure –cache_config –with-openssl

y en el fichero postgresql.conf:

ssl = true
```

- El servidor busca en $PGDATA los ficheros *server.key* (clave privada del servidor, que si está protegida en el arranque. Hay otro fichero *server.crt*
- Se usa el comando OpenSSL.

Generación de un certificado auto-firmado:

```
$ openssl req –new –text –out server.req    // pregunta muchas cosas

// si se desea que no pida la palabra de paso:

$ openssl rsa –in privkey.pem –out server.key
$ rm privkey.pem

// terminar, convertir el certificado X.509 en auto-firmado:

$ openssl req –x509 –in server.req –text –key server.key –out server.crt
$ chmod og-rwx
$ cp server.key $PGDATA
$ cp server.crt $PGDATA
```

11.2.4 Conexión remota sin encriptar usando TCP/IP (solo las no SSL)

```
TIPO        BASE DATOS  USUARIOS    DIRECCIÓN         MÉTODO
HOSTNOSSL   base_datos  usuario     direcciónCIDR     método autentic.  [opción]
```

- idéntico a HOST pero sólos se aceptan conexiones que no usen SSL, mientras que HOST admite conexiones SSL.

Por ejemplo:

```
# TYPE DATABASE USER IP-ADDRESS IP-MASK  METHOD
# A usuarios de la lista en archivo users,
# bases de datos dbl y db2,
# solo conexiones localhost, usando ident
host        dbl,db2     @users      127.0.0.1   255.255.255.255   ident

# Todos los usuarios en base de datos otrabd,
# red IP 192.168.4.0/24 SSL y no SSL,
# usando cifrado MD5, excepto 192.168.4.15
host        all         all   192.168.4.15 255.255.255.255  reject
host        otrabd      all   192.168.4.0 255.255.255.0     md5

# Juan en las bases de datos listadas en el archivo dbjuan,
# desde la IP de su workstation usando Kerberos 5 sabre SSL
hostssl     @dbjuan     juan  209.84.1.29 255.255.255.255   krb5
```

11.3 Seguridad en la base de datos

- Autorización de alto nivel en la creación de roles (*SUPERUSER, CREATEUSER, CREATEDB)*

- Utilización de las sentencias SQL *GRANT y REVOKE* para la gestión de los privilegios sobre los objetos de las bases de datos.

- Utilización del concepto de *pertenencia a rol* y *herencia de privilegios* para mejorar la eficiencia de la gestión de privilegios del sistema

Por ejemplo, si existe un role llamado TECNICO que tiene una serie de privilegios, y los usuarios Juan y Elena son miembros de este role, heredan los mismos permisos de este role.

La sentencia GRANT se puede aplicar:

- usuarios y roles
- tablas
- bases de datos y esquemas
- funciones
- lenguajes
- tablespaces

Ver el manual de referencia para más detalles.

Capítulo 12. Tareas administrativas: Copias de seguridad y recuperación

12.1 Introducción

Una vez el sistema está en marcha, el administrador de base de datos debe establecer una serie de tareas rutinarias para mantener el servidor en buenas condiciones, estas tareas se pueden resumir en las siguientes:

- Copias de seguridad (y pruebas de restauración de vez en cuando)
- Tarea de limpieza: *VACUUM*
- Tarea de reindexar: *REINDEX*
- Mantenimiento del fichero de seguimiento

Conviene automatizar estas tareas mediante el uso de herramientas del SO (como el cron, rotatelogs, etc) y/o scripts.

Se deben realizar rutinariamente, se debe fijar la frecuencia adecuada para cada una de ellas y comprobar su correcta ejecución.

12.2 Copias de seguridad

Las copias de seguridad son esenciales para las recuperaciones frente a fallos, se deben determinar las frecuencia conveniente, que será función del tamaño de los datos y de la frecuencia con que son modificados, lo cual marcará la *estrategia de copias de seguridad*.

Algunas herramientas usadas para las copias de seguridad nos permitirán además poder trasladar datos de unas bases de datos a otras.

Además hemos de asegurarnos de que los logs, ficheros WAL, estén en otro disco, o hacer copias más frecuentes de estos logs ya que en la versión 8 ya se pueden copiar automáticamente y usar en la recuperación.

Existen tres formas principales de realizar copias de seguridad:

- copia de seguridad de ficheros del SO
- volcado SQL usando las herramientas PostgreSQL: *pg_dump (pg_dumpall) y pg_restore.*
- volcado en línea y recuperación en el punto (*PITR*)

También se pueden usar los mandatos de SQL de PostgreSQL, "COPY TO / COPY FROM" para hacer copias de respaldo de tablas y posteriormente restaurarlas.

Hay que tener claro que lo que estamos tratando aquí es la copia de seguridad de las bases de datos y/o de los clusters, no estamos haciendo copia de seguridad del software instalado, del cual, conviene de vez en cuando hacer una copia de seguridad de los ficheros del SO.

12.3 Copias de seguridad de ficheros del SO

Es el método más sencillo, pero el más ineficaz, se trata de realizar una copia de **todos** los ficheros de un *cluster*, por ejemplo:

```
$ su – postgres
$ cd /tmp/backup
$ tar cvfz copia.tar.gz $PGDATA
$ --- > copiar el fichero anterior a cinta …
```

Desventajas de este método:

- la base de datos debe estar parada
- no se pueden recuperar partes del cluster (bases de datos, esquemas, tablas, etc.)

La recuperación consiste en borrar todo el cluster y descomprimir el fichero de copia de seguridad, con lo que se pierden los datos que se hayan modificado desde la última copia de la base de datos.

12.4 Volcado SQL

Los volcados de este tipo se realizan usando las herramientas que nos proporciona PostgreSQL. Estos volcados son muy flexibles y de gran utilidad, nos permitirán hacer copias de seguridad de toda la base de datos o de partes de ella, y luego, dada una copia de seguridad, nos permitirán restaurar lo que queramos.

Además, estas herramientas sirven para la transmisión de datos entre bases de datos.

Las herramientas que se van a utilizar son:

pg_dump: vuelca una base de datos o parte de ella a un fichero, bien en texto plano o en un formato propio de PostgreSQL. Se puede recuperar cualquier objeto que esté en el fichero aisladamente. El servidor debe estar en marcha.

pg_dumpall: vuelca un cluster completo

pg_restore*:* recupera los objetos volcados en una copia de seguridad que no se realizón en texto plano sino en un formato propio de PostgreSQL.

psql : se usa para recuperar los volcados en texto plano.

12.4.1 pg_dump

Con este comando, podemos volcar una base de datos o parte de ella a un fichero script en texto plano o en un formato propio de PostgreSQL.

Si lo hacemos en un fichero de texto plano:

- tendremos un fichero de texto con instrucciones SQL
- podemos usar *psql* para restaurar
- es portable a servidores SQL

Si lo hacemos en un formato propio de PostgreSQL:

- debemos usar *pg_restore* para restaurar
- es más flexible
- sólo es portable entre servidores PostgreSQL (aunque esto no es del todo cierto, debido a que con *pg_restore* se pueden crear ficheros SQL en texto plano).

Este proceso se debe realizar con el cluster en marcha, pero no bloquea la base de datos (ni lectores ni escritores). Es un sistema análogo al *export* de Oracle.

```
Uso:
  pg_dump [OPCIÓN]... [NOMBREDB]

Opciones generales:
  -f, --file=ARCHIVO       nombre del archivo de salida
  -F, --format=c|t|p       Formato del archivo de salida
                           (personalizado, tar, sólo texto)
  -i, --ignore-version     procede aún cuando las versiones del servidor
                           y pg_dump no coinciden
  -v, --verbose            modo verboso
  -Z, --compress=0-9       nivel de compresión para formatos comprimidos
  --help                   muestra esta ayuda y termina
  --version                muestra información de la versión y termina

Opciones que controlan el contenido de la salida:
  -a, --data-only          extrae sólo los datos, no el esquema
  -c, --clean              limpia (tira) el esquema antes de su creación
  -C, --create             incluye comandos para crear la base de datos
                           en la extracción
  -d, --inserts            extrae los datos usando INSERT, en vez de COPY
  -D, --column-inserts     extrae los datos usando INSERT con nombres
                           de columnas
  -E, --encoding=CODIFIC  extrae los datos con la codificación CODIFIC
  -n, --schema=ESQUEMA     extrae sólo el esquema nombrado
  -o, --oids               incluye OIDs en la extracción
  -O, --no-owner           en formato de sólo texto, no reestablecer
                           los dueños de los objetos
  -s, --schema-only        extrae sólo el esquema, no los datos
  -S, --superuser=NAME     especifica el nombre del superusuario a usar en
                           el formato de sólo texto
  -t, --table=TABLE        extrae sólo la tabla nombrada
  -x, --no-privileges      no extrae los privilegios (grant/revoke)
  -X disable-dollar-quoting, --disable-dollar-quoting
                           deshabilita el uso de delimitadores de «dólar»
```

```
                           usa delimitadores de cadena estándares
  -X disable-triggers, --disable-triggers
                           deshabilita los disparadores (triggers) durante el
                           restablecimiento de la extracción de sólo-datos
  -X use-set-session-authorization, --use-set-session-authorization
                           usa comandos SESSION AUTHORIZATION en lugar de
                           comandos OWNER TO
Opciones de la conexión:
  -h, --host=ANFITRIÓN     anfitrión de la base de datos o el
                           directorio del enchufe (socket)
  -p, --port=PUERTO        número del puerto de la base de datos
  -U, --username=USUARIO   nombre de usuario con el cual conectarse
  -W, --password           fuerza un prompt para la contraseña
                           (debería ser automático)
```

Si no se especifica un nombre de base de datos entonces el valor de la variable de ambiente `PGDATABASE` es usado, y lo mismo con `PGUSER, PGHOST y PGPORT.`

Tiene más opciones, usando *'$ man pg_dump'* se pueden ver detalles de esas opciones.

Comentamos aquí algunos parámetros:

- Formato del fichero de volcado: `-F, --format=c|t|p`
  - p: valor por defecto, texto plano
  - t: fichero *tar* adecuado para *pg_restore.* Este formato permite reordenaciones, y/o exclusiones de elementos del esquema cuando se recupera la base de datos. También es posible limitar qué datos se recuperan.
  - c: archivo comprimido adecuado para *pg_restore*. Es el formato más flexible.

- Uso de *INSERT* en vez de *COPY*: `-d, --inserts`
  - COPY es más rápido.
  - INSERT es más portable.

- Especificación del fichero de volcado: `-f fichero, --file=fichero`
  - si no se especifica, genera la salida por *stdout,* con lo que deberíamos usar redirecciones para que el resultado quedara en un fichero.

Como se ve también, se pueden volcar esquemas, tablas, sólo datos o sólo las estructuras.

Ejemplos de uso:

```
-- Exportar una base de datos:

$ pg_dump –f bdnominas.sql –h otrohost –p 6432 bdnominas

-- Exportar un esquema, redireccionando la salida

$ pg_dump –n nominas2004 > esq_nominas2004.sql

-- Volcar una base de datos llamada mibd en un fichero tar:
$ pg_dump -Ft mibd > bd.tar
```

12.4.2 pg_dumpall

Con *pg_dumpall* se realizan volcados del cluster completo, incluyendo *roles de grupo y roles de login.*

```
Uso:

  pg_dumpall [OPCIÓN]...


Opciones generales:
  -i, --ignore-version     procede aún cuando la versión del servidor y
                           pg_dumpall no coinciden
  --help                   muestra esta ayuda y termina
  --version                muestra información de la versión y termina

Opciones que controlan el contenido de la salida:
  -a, --data-only          extrae sólo los datos, no el esquema
  -c, --clean              tira la base de datos antes de crearla
  -d, --inserts            extrae los datos usando INSERT, en vez de COPY
  -D, --column-inserts     extrae los datos usando INSERT con nombres
                           de columnas
  -g, --globals-only       extrae sólo los objetos globales, no bases de datos
  -o, --oids               incluye OIDs en la extracción
  -O, --no-owner           no reestablece los dueños de los objetos
  -s, --schema-only        extrae sólo el esquema, no los datos
  -S, --superuser=NAME     especifica el nombre del superusuario a usar en
                           el guión
  -x, --no-privileges      no extrae los privilegios (grant/revoke)
  -X disable-dollar-quoting, --disable-dollar-quoting
                           deshabilita el uso de delimitadores de «dólar»
                           usa delimitadores de cadena estándares
  -X disable-triggers, --disable-triggers
                           deshabilita los disparadores (triggers) durante el
                           restablecimiento de la extracción de sólo-datos
  -X use-set-session-authorization, --use-set-session-authorization
                           usa comandos SESSION AUTHORIZATION en lugar de
                           comandos OWNER TO

Opciones de la conexión:
  -h, --host=ANFITRIÓN     anfitrión de la base de datos o el
                           directorio del enchufe (socket)
  -p, --port=PUERTO        número del puerto de la base de datos
  -U, --username=USUARIO   nombre de usuario con el cual conectarse
  -W, --password           fuerza un prompt para la contraseña
                           (debería ser automático)
```

Tiene más opciones, usando *'$ man pg_dumpall'* se pueden ver detalles de esas opciones.

Como se ve, este comando es similar a *pg_dump,* pero tiene algunas diferencias:

- no permite la opción de indicar el fichero de salida, con lo cual hay que redireccionar la salida a un fichero.
- no tiene la opción de formatos de fichero de salida, la salida es siempre un fichero de texto plano.
- tiene una opción muy interesante que es *"-g"* que permite exportar únicamente los objetos globales, con lo que podemos generar un script que contenga la creación de usuarios y roles.

Ejemplos de uso:

```
-- Exportar un cluster:
$ pg_dumpall > micluster.sql

-- Exportar solo los datos
$ pg_dumpall –a > misdatos.sql

-- Exportar solo los roles / usuarios de una base de datos en otro servidor
-- que además, tiene otro superusuario
$ pg_dump_all –g –h otroservidor –U otrosuperusuario –p 6432 > misusuarios.sql

-- Exportar solo los esquemas
$ pg_dumpall –s > misesquemas.sql
```

12.4.3 Recuperación con psql

La recuperación con *psql* se puede hacer cuando los ficheros de volcado son texto plano con sentencias SQL. Hacemos uso de la posibilidad que tiene *psql* de ejecutar código que le viene redireccionado de entrada desde un fichero.

```
$ psql [bd_destino] < fichero.sql
```

El fichero se puede modificar antes de cargarlo, pero hay que llevar mucho cuidado con los editores que gastemos, porque hay algunos que realizan conversiones de codificación de caracteres o formatos y esto puede producir efectos indeseados.

Si lo que se quiere recuperar es un cluster completo o la parte global (roles/usuarios), siempre se debe restaurar en la base de datos *template1*.

Para ilustrar todo esto, vamos a imaginarnos que queremos hacer copias de seguridad de un servidor a otro con los siguientes datos:

	servidor origen	**servidor destino**
host	origen[.gva.es]	destino[.gva.es]
puerto	5432	6432
superusuario	postgres	spdestino
base de datos	bdorigen	bddestino
tabla	empleados	empleados

y que todas las operaciones las hacemos desde un pc, que tiene instalado un cliente de PostgreSQL compatible con las versiones de los servidores, así ilustramos el uso de los parámetros de conexión, y estoy conectado en el pc con el usuario *postgres:*.

```
-- crear los usuarios de un cluster en otro, atención porque en el segundo paso
-- nos conectamos a template1:

$ pg_dumpall –h origen –U postgres –g > usuarios_cluster.sql
$ psql –h destino –p 6432 –U spdestino template1 < usuarios_cluster

-- copiar una base de datos en el otro servidor

$ pg_dump –h origen –U postgres –f bdorigen.sql –C bdorigen
```

```
$ psql –h destino –p 6432 –U spdestino bddestino < bdorigen.sql

-- copiar los objetos de una base de datos en otra (la diferencia con el método
-- anterior es que en el pg_dump no ponemos '-C' y que antes, aunque nos
-- conectamos a bddestino en el psql, crea una nueva base de datos llamada
-- bdorigen

$ pg_dump –h origen –U postgres –f bdorigen.sql bdorigen
$ psql –h destino –p 6432 –U spdestino bddestino < bdorigen.sql

-- finalmente, un ejemplo de cómo la salida de un comando se puede enviar a otro
-- ademas de poder usar variables de entorno en el comando, simplemente vamos a
-- copiar una tabla de una bd a otra:

$ export PGUSER=postgres
$ export PGDATABASE=bdorigen
$ PGPORT=5432 PGHOST=origen pg_dump –t empleados –f empleados.sql
$ export PGUSER=spdestino
$ export PGDATABASE=bddestino
$ PGPORT=6432 PGHOST=destino psql < empleados.sql > empleados.log

-- por último, si ejecutamos desde dentro de 'origen', podríamos hacer que la
-- salida de un pg_dump pasara por una 'tubería' al proceso de restauración:

$ pg_dump bdorigen | ssh destino psql –U spdestino bddestino
```

12.4.4 Recuperación con pg_restore

Con *pg_restore* se pueden restaurar a partir de ficheros en formato distinto a texto plano (*tar* o uno propio de PostgreSQL comprimido). Se puede seleccionar y reordenar la selección.

```
Uso:

  pg_restore [OPCIÓN]... [ARCHIVO]

Opciones generales:
  -d, --dbname=NOMBRE      nombre de la base de datos a la que conectarse
  -f, --file=ARCHIVO       nombre del archivo de salida
  -F, --format=c|t         especifica el formato del respaldo
  -i, --ignore-version     procede aún cuando la versión del servidor
                           no coincide
  -l, --list               imprime una tabla resumida de contenidos del archivador

  -v, --verbose            modo verboso
  --help                   muestra esta ayuda y termina
  --version                muestra información de la versión y termina

Opciones que controlan la recuperación:
  -a, --data-only          reestablece sólo los datos, no el esquema
  -c, --clean              limpia (tira) el esquema antes de su creación
  -C, --create             crea la base de datos de destino
  -I, --index=NOMBRE       reestablece el índice nombrado
  -L, --use-list=ARCHIVO   usa la tabla de contenido especificada para ordenar
                           la salida de este archivo
  -n, --schema=NAME        reestablece el esquema únicamente, no los datos
  -O, --no-owner           no reestablece los dueños de los objetos
  -P, --function=NOMBRE(args)
                           reestablece la función nombrada
  -s, --schema-only        reestablece el esquema únicamente, no los datos
  -S, --superuser=NOMBRE   especifica el nombre del superusuario que se usa
```

```
                            para deshabilitar los disparadores (triggers)
  -t, --table=NOMBRE        reestablece la tabla nombrada
  -T, --trigger=NOMBRE      reestablece el disparador (trigger) nombrado
  -x, --no-privileges       no reestablece los privilegios (grant/revoke)
  -X disable-triggers, --disable-triggers
                            deshabilita los disparadores (triggers) durante el
                            restablecimiento de la extracción de sólo-datos
  -X use-set-session-authorization, --use-set-session-authorization
                            usa comandos SESSION AUTHORIZATION en lugar de
                            comandos OWNER TO

Opciones de la conexión:
  -h, --host=ANFITRIÓN      anfitrión de la base de datos o el
                            directorio del enchufe (socket)
  -p, --port=PUERTO         número del puerto de la base de datos
  -U, --username=USUARIO    nombre de usuario con el cual conectarse
  -W, --password            fuerza un prompt para la contraseña
                            (debería ser automático)
  -e, --exit-on-error       abandonar al encontrar un error
                            por omisión, se continúa la restauración

Si no se proporciona un nombre de archivo de salida, se usa la salida estándar.
```

Tiene más opciones, usando *'$ man pg_restore'* se pueden ver detalles de esas opciones.

Hay dos modos de hacerlo:

- directamente a una base de datos, es la opción por defecto '-d'
- volcando en otro fichero que actúa de conversor. Se crea así un script en texto plano que podemos modificar antes de ejecutarlo.

Por ejemplo, supongamos que hemos hecho una copia de seguridad de nuestra base de datos en formato comprimido y se han perdido datos de una tabla de empleados, para recuperarlos hacemos:

```
$ pg_restore -Fc -f empleados.sql -t empleados bd.dump
$ pgsql bdorigen –c 'drop table empleados'
$ psql bdorigen < empleados.sql
```

12.4.5 Recuperación ante fallos

Fallos en la memoria principal

PostgreSQL se recupera bien en fallos de la memoria primaria (cortes de luz ,cuelgues, etc.), el sistema se recupera usando los ficheros de log (WAL) y nunca se pierde una transacción confirmada.

Los ficheros de log permiten que además, la base de datos no tenga que estar completamente actualizada (*sync)* para mantener los datos ante fallos en la memoria principal (pérdida de valores en los bufferes compartidos).

Hay un número máximo de logs: 2*CHECKPOINT_SEGMENTS+1

Si se llenan sin que se haya hecho *sync,*el sistema hace un *sync* y avisa de que hacen falta más logs.

El trabajo y el tiempo de recuperación dependen del número de checkpoint_segments, cuanto mayor sea el sistema aguantará más tiempo sin llevar los búferes sucios a disco y será más eficiente, aunque también se requiere más espacio en disco y las recuperaciones requeriran más tiempo.

Fallos en memoria secundaria (disco)

Hasta la versión 7.X no había recuperación total, si se producía una pérdida dentro del $PGDATA, había que recurrir a la copia de seguridad física más reciente y desde ese momento las actualizaciones se perdían, aunque tuviéramos los ficheros de log. Aquí es donde, en la versión 8.X se incorpora el volcado en línea y la recuperación PITR

12.5 Volcado en linea y recuperación PITR

A partir de la versión 8.0.0, PostgreSQL permite actualizar la copia de seguridad con los cambios que proporcionan en los ficheros de log que hayan sobrevivido.

Para poder utilizar esta opción tiene que estar habilitado el archivado WAL y en funcionamiento. Los ficheros WAL son segmentos de 16Mb que se nombran secuencialmente en el cual el sistema va copiando los cambios en la base de datos. El sistema recicla los ficheros de log que no van a ser necesitados renombrándolos a números superiores dentro de la secuencia.

Para activar el archivado, en el fichero *postgresql.conf* debemos indicar el comando de copia para preservar los ficheros de log, parámetro *archive_command,* haciendo, por ejemplo:

```
archive_command = ‘cp –i %p /mnt/server/archivedir/%f < /dev/null’
```

donde ‘%p’ representa el nombre del fichero con la ruta absoluta y ‘%f’ sin la ruta.

Para realizar ***copias de seguridad en línea*** se siguen los siguientes pasos:

1. activar el archivado WAL
2. antes de empezar y desde una consola SQL hay que ejecutar:

   ```
   select pg_start_backup(‘nombre_copia’);
   ```

3. con el servidor en marcha, hacemos la copia desde el sistema operativo, no hace falta parar el servidor, por ejemplo:

   ```
   $ tar –cvf backup_nombre_copia.tar $PGDATA
   ```

4. cuando acaba, desde la consola SQL, marcamos el final, ejecutamos:

   ```
   select pg_stop_backup();
   ```

5. se crea así un fichero de marca en el directorio $PGDATA/pg_xlog/archive_status y copia los logs que se reciclan en donde se haya indicado en *archive_command.*

Para realizar una **recuperación** se siguen los siguientes pasos:

1. parar el postmaster
2. si tenemos espacio, copiar el cluster dañado y los tablespaces a otra ubicación
3. borrar todos los ficheros que hay dentro del cluster asi como los correspondientes a los tablespaces
4. recuperar la copia de seguridad (comprobando los tablespaces en *pg_tblspc*)

   ```
   $ tar xvf backup_nombre_copia
   ```

5. borrar los ficheros WAL en $PGDATA/pg_xlog porque probablemente estarán obsoletos.
6. si existen ficheros WAL sin archivar, tal como hemos hecho en el paso 2, copiarlos a *pg_xlog*
7. crear un fichero de comandos de recuperación, *recovery.conf* en el directorio $PGDATA. Existe una plantilla, *recovery.conf.sample,* en el directorio *$HOME/pgsql/share.* Se copia con el nombre *"recovery.conf"* en $PGDATA y se edita. En principio, si se quiere recuperar todo y los logs están en su sitio, no hay que tocar nada.
8. se arranca PostgreSQL que entra en modo de recuperación y procede a leer la información de los WAL que necesite, siendo necesario evitar que se conecten usuarios normales durante la recuperación.
9. inspeccionar la base de datos, si todo ha ido correcto, los datos estarán recuperados hasta la última transacción confirmada y el fichero *recovery.conf* se renombra a *recovery.done.*

Conviene tener los ficheros log en un disco distinto al que está el cluster, para que la recuperación sea mejor. Para ello, debemos mover el directorio *pg_xlog* a otro disco y crear un enlace simbólico, con el cluster parado, por ejemplo:

```
$ mkdir /disco2/pg/
$ cd $PGDATA
$ mv pg_xlog /disco2/pg
$ ln –s /disco2/pg/pg_xlog pg_xlog
```

además, para asegurarnos que no se nos pierden archivos de log, podemos hacer un cron que copie los archivos que no están llenos en una ubicación distinta.

En el fichero *recovery_conf* hay una serie de parámetros que pueden ayudar a recuperar hasta el momento o la transacción que queramos (lo que se conoce como **recuperación Point-in-time**):

- *restore_command*: lo que ejecuta esta variable es lo que PostgreSQL ejecutará antes de empezar la recuperación. Por ejemplo:

  ```
  restore_command = 'cp /mnt/server/archivedir/%f %p'
  ```

- *recovery_target_time*: hasta qué momento
- *recovery_target_xid*: hasta una transacción determinada
- *recovery_target_inclusive:* si los dos casos anteriores son inclusive o no.

12.6 Resumen: actuación ante fallos:

Fallo	Acciones
Caída del cliente	Nada, las transacciones en progreso se deshacen
Caída grácil del servidor	Nada, las transacciones en progreso se deshacen
Caída abrupta del servidor	Nada, las transacciones en progreso se deshacen y las finalizadas se pasan de los logs a los datos.
Caída del SO	Nada, las transacciones en progreso se deshacen y las finalizadas se revisan. Las escrituras parciales de páginas se reparan.
Caída del disco de datos pero tenemos los logs y copia de seguridad	recuperación desde la última copia de seguridad y posterior uso de recovery.conf para recuperar las transacciones hasta el momento del fallo.
Caída de disco de logs	Nada que hacer desde PostgreSQL. Si algún log parcial se ha perdido, se pierden las transacciones desde el último sync (manual checkpoint o automático),
Borrado de una tabla importante con transacción confirmada	Dos opciones: 1. recuperación de copia de seguridad y de las transacciones hasta que se borró la tabla (point-in-time) 2. recuperación de la copia de seguridad en otro cluster, hacer pg_dump y luego pg_restoreen el cluster original
Borrado accidental de un fichero en $PGDATA/global	Dos opciones: 1. recuperar de backup, lo normal es que no haya problema y que el sistema no quede inconsistente. 2. identificar qué almacenaba el fichero perdido y recuperarlo con pg_dump y pg_restore.

Capítulo 13. Tareas administrativas: Vacuum

Vacuum es un proceso de limpieza que se encarga de:

- recuperar el espacio de disco ocupado por filas modificadas o borradas.
- actualizar las estadísticas usadas por el planificador / optimizador
- evitar la pérdida de datos muy antiguos debido al reuso del identificador de transacción

La frecuencia con que se ejecute este proceso depende de cada instalación y, además, se puede ejecutar en paralelo con operaciones de actualización pero no de definición, se puede ejecutar desde el SO o desde SQL. Un esquema de lo que hace VACUUM es:

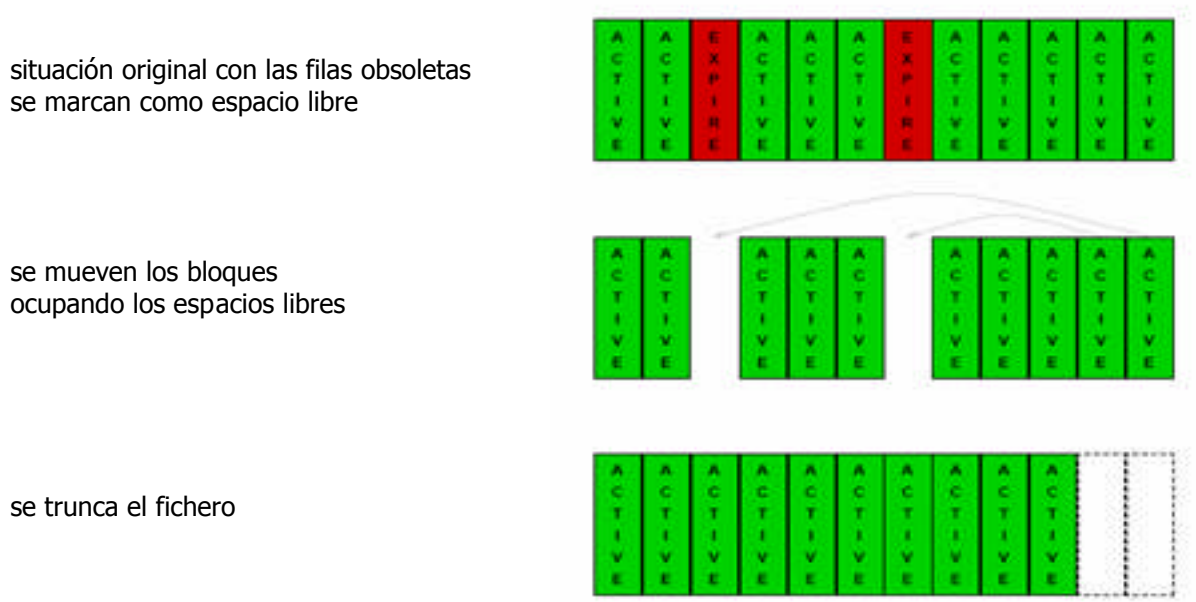

Figura 13-1: Funcionamiento de Vacuum

13.1 Vacuum desde SO:

```
  vacuumdb [OPCIÓN]... [BASE-DE-DATOS]

Opciones:
  -a, --all                 limpia todas las bases de datos
  -d, --dbname=BASE         base de datos a limpiar
  -t, --table='TABLE[(COLUMNS)]'
                            limpiar sólo esta tabla
  -f, --full                usar «vacuum full»
  -z, --analyze             actualizar las estadísticas
  -e, --echo                mostrar los comandos enviados al servidor
  -q, --quiet               no desplegar mensajes
  -v, --verbose             desplegar varios mensajes informativos
  --help                    mostrar esta ayuda y salir
  --version                 mostrar el número de versión y salir

Opciones de conexión:
  -h, --host=ANFITRIÓN      nombre del servidor o directorio del socket
  -p, --port=PUERTO         puerto del servidor
  -U, --username=USUARIO    nombre de usuario para la conexión
  -W, --password            preguntar la contraseña
```

13.2 Vacuum de SQL:

```
VACUUM [ FULL | FREEZE ] [ VERBOSE ] [ table ]
VACUUM [ FULL | FREEZE ] [ VERBOSE ] ANALYZE [ table [ (column [, ...] ) ] ]
```

para obtener mayores detalles *"man vacuum"*.

- desde el SO se puede hacer un VACUUM de todas las bases de datos del cluster.
- sin tabla: se realiza para toda la base de datos
- sin FULL: sólo reclama espacio y lo deja libre para nuevas filas, lo agrega al FSM (Free Space Map)
- FULL: mueve filas entre bloques para compactar la tabla. Se exige bloquear exclusivamente la tabla.
- ANALYZE: es una limpieza junto con una actualización de las estadísticas.
- FREEZE: tarea de bajo nivel, se hace para evitar el problema del reuso del identificador de transacción, sólo se hace en bases de datos que se limpiaron hace mucho tiempo.

13.3 Recomendaciones:

- usar VACUUM FULL si alguna tabla ha cambiado mucho y no va a crecer en el futuro.
- ejecutar VACUUM diariamente en todas las bases de datos
- hacerlo con más frecuencia sobre tablas con mucha carga de actualización.
- determinar las tablas grandes con pocas actualizaciones para eliminarlas del VACUUM.

Dentro del VACUUM, se pueden actualizar las estadísticas con la opción ANALYZE, haciendo lo mismo que hace el comando ANALYZE, así proporciona:

- información usada por el optimizador
- no es muy pesado el proceso porque se realiza sobre una muestra al azar.
- necesario si la distribución de los datos cambia mucho

13.4 El problema del XID (identificador de transacción)

Otro problema que resuelve VACUUM es el evitar la pérdida de datos muy antiguos, porque MVCC introduce el concepto de marca de tiempo **XID identificador de transacción.**

XID está limitado a 32 bits, es decir cuando llega a 2^{32} se reinicializa, tiene un funcionamiento circular, si esto llega a ocurrir, se perderían datos. Existen además dos XID especiales: *BootstrapXID y FrozenXID.* Cada vez que se ejecuta VACUUM se guarda información que se puede consultar:

```
select datname, age(datfrozenxid) from pg_database;
```

así, si se ejecuta poco el vacuum, la base de datos avisa al administrador, se deben limpiar las tablas una vez cada mil millones de transacciones. El caso es que se recomienda hacer VACUUM diario.

13.5 AUTOVACUUM

Para evitar los problemas con la ejecución de VACUUM, a partir de la versión 8, PostgreSQL incorpora un proceso de fondo **AUTOVACUUM** que se ejecuta periódicamente cuando está habilitado y él mismo comprueba las tablas con alta carga de actualizaciones. Hay una serie de parámetros en *postgresql.conf* para modificar el comportamiento de este proceso.

Capítulo 14. Reindex

Otras tareas importantes son las reindexaciones, ya que pueden haber índices incorrectos por problemas en el software o hardware o porque haya que optimizar el índice debido a que tiene páginas *muertas* que hay que eliminar.

Desde el SO:

```
reindexdb [connection-option...] [--table | -t table ] [--index | -i index ]
[dbname]
reindexdb [connection-option...] [--all | -a]
reindexdb [connection-option...] [--system | -s] [dbname]
```

Desde SQL:

```
REINDEX { INDEX | TABLE | DATABASE | SYSTEM } name [ FORCE ]
```

se pueden reindexar las bases de datos completas, tablas, índices o el catálogo del sistema.

Este operador bloquea las tablas en modo exclusivo cuando las reindexa.

Ejemplos:

```
-- reparar un indice:
reindex index nomindice;

-- reparar todos los índices de una tabla:
reindex table nomtabla;

-- reparar los índices del catálogo de base de datos, para esto
-- hay que levantar postmaster sin que use los índices y luego reindexar
$ export PGOPTIONS="-P"   // significa lo mismo que $ postmaster –P
$ psql bd
bd=# reindex database bd;

-- reparar los índices del catálogo compartido (pg database, pg group, pg shadow,
-- etc.). No se debe levantar postmaster si no que hay que arrancar un proceso
-- autónomo:
$ postgres –D $PGDATA –P prueba
backend> reindex index indice_catalogo_compartido
```

Se puede introducir en el cron que realice ciertas operaciones de mantenimiento:

```
$ crontab –e
// se abre el fichero de cron y añadimos estas líneas:
0 3 * * * psql -c 'VACUUM FULL;' test
0 3 * * * vacuumdb -a –f
0 30 * * * reindexdb bd
```

¡ATENCIÓN!, problema entre Reindex y Archivado WAL:

Hay que tener presente que cuando reindexamos, se rehacen muchos ficheros, con lo que hay que llevar especial cuidado si el *archivado WAL* está activado, pues una reindexación de toda la base de datos implica que **se puede duplicar el tamaño de los ficheros de la base de datos**, ya que el sistema de WAL crea archivos de log para cada cambio

.

Capítulo 15. Mantenimiento fichero de seguimiento

PostgreSQL al arrancar va informando de todo aquello que considera relevante para que se pueda hacer un seguimiento. Hay varias formas de hacer este seguimiento:

- en la salida estandar: incorrecto porque se pierde mucha información
- a un fichero, se puede rotar, pero hay que parar el servidor para hacerlo.
- al *syslog* del sistema, con el parámetro *syslog=2* en postmaster.conf y se puede rotar usando los mecanismos que da el sistema.

Si queremos que vaya a un fichero, hacemos:

```
$ pg_ctl [otras opciones] –l $PGLOG start
```

donde $PGLOG apunta a un archivo.

Este archivo tiene informacion esencial para ver qué ocurre en caso de funcionamiento raro. En el fichero *postgresql.conf* hay una serie de parámetros que afectan al fichero y puede adoptar un tamaño grande con altos nivels de seguimiento.

Como este fichero se hace muy grande, se plantea la necesidad de “rotarlo”. Este concepto consiste en que cada cierto tiempo, el fichero de log se copia a otro fichero con otro nombre y se deja vacío el original. Esta función se puede configurar en el fichero postgresql.conf o bien usar programas que vienen en el SO o en Apache, como *logrotate o rotatelogs.*

Un truco para hacer esta rotación sería este. Suponiendo que todos los días, en un momento determinado apagamos el cluster porque vamos a hacer una copia física, al terminar ésta tenemos que arrancar el cluster, lo podríamos hacer de este modo:

```
$ pg_ctl –D $PGDATA –l serverlog.`date +%j.%H.%M` start
```

donde el comando ‘date’ está rodeado entre “acentos”, para provocar que la shell ejecute el comando, de este modo, nos devuelve una cadena que tiene 3 números, el primero el día del año, el segundo es la hora del día y el tercero los minutos.

Capítulo 16. Catálogo del sistema

El ***catálogo del sistema*** consiste en un conjunto de objetos de esquema (tablas y vistas) que permiten conocer el contenido de la base de datos.

Son un conjunto de tablas y vistas cuyo acceso puede estar restringido (si la información que contiene es privada) y sólo son accesibles por el superusuario, como *pg_authid*, y no todas se actualizan automáticamente, por ejemplo, algunas columnas de *pg_class* sólo se actualizan cuando se ejecuta *vacuum o analyze,* o si no se activan las estadísticas- Todo esto es importante tenerlo en cuenta cuando se realicen monitorizaciones.

Existen tablas que son compartidas por todas las bases de datos del cluster, como *pg_database, pg_shadow o pg_group* y otras que no se comparten y están cada una en su base de datos. Las podemos encontrar dentro de los esquemas *pg_catalog y pg_toast* de cada base de datos.

Son difíciles de usar, se plantean algunas alternativas:

- uso de herramientas que facilitan la labor: *pgadmin o pgaccess.*
- *The orapgsqlviews Project (Oracle Style Data Dictionary views for PostgreSQL)*: es un proyecto que quiere proporcionar un catálogo similar al de Oracle y en segundo lugar otras vistas más cómodas y autoexplicativas (http://gborg.postgresql.org).

16.1 Tablas

NOMBRE	USO
pg_aggregate	funciones agregadas
pg_am	métodos de acceso a índices
pg_amop	operadores asociados con las clases de los operadores de métodos de acceso a índices
pg_amproc	procedimientos soportados asociados con las clases de los operadores de métodos de acceso a índices
pg_attrdef	valores por defecto de las columnas
pg_attribute	información sobre las columnas de las tablas; una fila por cada columna de cada tabla
pg_authid	almacena los roles de la base de datos
pg_auth_members	matiene las relaciones de pertenencia entre los roles
pg_autovacum	almacena parámetros de configuración para el demonio del autovacum
pg_cast	almacena métodos de conversión de tipos de datos, tanto los predefinidos como los definidos por el usuario con CREATE CAST

NOMBRE	**USO**
pg_class	almacena información sobre los objetos (tablas, índices, secuencias, vistas y otros objetos especiales), una fila por cada tabla, índice, secuencia o vista.
pg_constraint	almacena la definición de las restricciones *check,* clave primaria, unicidad, y clave ajena de las tablas; las restricciones de valor no nulo se - almacenan en *pg attribute*
pg_conversion	información sobre las conversiones de codificación (juegos de caracteres)
pg_database	bases de datos creadas en el *cluster*
pg_depend	guarda las relaciones de dependencia entre objetos de la base de datos (permite implementar el DROP ... CASCADE o DROP ... RESTRICT)
pg_description	almacena una descripción o comentario opcional sobre los objetos
pg_index	información adicional sobre los índices
pg_inherits	jerarquía de la herencia de las tablas
pg_language	interfaces de llamadas o lenguajes en los que se pueden escribir los procedimientos almacenados
pg_largeobject	guarda la información que maquilla a los LOB; un LOB es identificado por un OID que se le asigna cuando se crea; los LOB se almacenan divididos en trozos pequeños
pg_listener	da soporte a las instrucciones LISTEN y NOTIFY; una *listener* crea una entrada para cada notificación que está esperando; un notificador rastrea *pg listener* y actualiza cada entrada que le casa para informar que se ha producido una notificación
pg_namespace	esquema y propietarios de esquemas
pg_opclass	guarda las clases de operadores de métodos de acceso. a índices
pg_operator	información sobre los operadores definidos, ya sean predefinidos o definidos por el usuario
pg_proc	información sobre funciones y procedimientos
pg_rewrite	almacena las reglas de reescritura definidas
pg_statistic	información estadística; es mejor usa las vistas para visualizarla
pg_tablespace	información sobre los *tablespaces* del cluster
pg_trigger	información sobre la definición de los disparadores creados
pg_type	almacena información sobre los tipos de datos ya sean escalares (CREATE TYPE), tipos compuestos (para cada tabla o fila de tabla),

16.2 Vistas

NOMBRE	USO
pg_group	compabilidad hacia atrás; muestra los roles que no son de conexión
pg_indexes	información útil de los índices
pg_locks	muestra los bloqueos que se están llevando acabo para la ejecución de las transacciones en curso
pg_prepared_xacts	información sobre las transacciones preparadas para el protocolo de confirmación en dos fases
pg_roles	información útil sobre los roles (más legible que *pg authid)*
pg_rules	información sobre las reglas de reescritura creadas
pg_settings	información sobre los parámetros de puesta en marcha del servidor
pg_shadow	compatibilidad hacia atrás; muestra los roles de conexión
pg_stats	información legible de *pg statistics*
pg_tables	información legible sobre las tablas creadas
pg_user	información sobre los usuarios; forma alternativa de *pg shadow*
pg_views	información sobre las vistas creadas en la base de datos

Capítulo 17. Monitorizacion

La monitorización de la actividad del servidor PostgreSQL se puede hacer con herramientas del SO o con herramientas propias de la base de datos.

17.1 Monitorización de la actividad en la Base de Datos

17.1.1 Comandos del Sistema Operativo (Linux / Unix)

La monitorización se debe fijar sobre todo en:

- Uso de swap (free, vmstat, etc.)
- Uso del disco (iostat, etc.)
- Monitoreo interactivo (top, ps)

ps

Usando el comando *ps*, podemos filtrar las filas en las que salga *"postmaster:"*

```
-- ver el proceso servidor:
$ ps -ef |grep "postmaster" | grep -v grep
-- ver los procesos backend
$ ps -ef |grep "postgres:" | grep -v grep
```

supongamos que tenemos una salida como la siguiente:

```
$ ps –ef | grep postmaster | grep –v grep
27955 pts/1    S      0:00 /usr/local/alvh/pgsql/bin/postmaster
27956 pts/1    S      0:00 postgres: stats buffer process
27957 pts/1    S      0:00 postgres: stats collector process
27960 pts/1    S      0:00 postgres: alvh alvh 127.0.0.1 SELECT waiting
27962 pts/1    S      0:00 postgres: alvh alvh [local] idle in transaction
```

Nos salen los siguientes procesos:

- **procesos de fondo:**
  - *postmaster,*
  - pueden salir también "*postgres: logger process" y "postgres: writer process".*
- **procesos del recolector de estadísticas:**
  - *postgres: stats buffer process*
  - *postgres: stats collector process*

- **procesos servidores desde clientes:**
  - *postgres: <usuario> <basedatos> <host> <actividad>*

  <actividad> puede ser:

  - *idle*: el proceso está inactivo
  - *idle in transaction*: el proceso está inactivo en medio de una transacción
  - *waiting*: bloqueado, en espera del resultado de otra transacción
  - *comando*: tipo de comando

Todo esto lo podemos hacer desde SQL, si activamos el parámetro "*stats_command_string = on"*

```
$ pg_ctl reload
$ psql test
test=> SELECT * FROM pg_stat_activity;

datid | datname | procpid | usesysid | usename  | current_query
------+---------+---------+----------+----------+-----------------------------
16570 | test    | 27826   | 1        | postgres | select * from pg_class;
16570 | test    | 27844   | 1        | postgres | <IDLE>
(2 rows)
```

top

Información de las tareas ejecutadas en el sistema.

```
$ top
```

```
top - 23:20:26 up 20:28,  1 user,  load average: 0.08, 0.02, 0.01
Tasks:  34 total,   2 running,  32 sleeping,   0 stopped,   0 zombie
Cpu(s):  51.8% user,  28.1% system,   0.0% nice,  20.1% idle
Mem:    159796k total,   155844k used,     3952k free,    12388k buffers
Swap:   128480k total,        0k used,   128480k free,    97336k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
24610 postgres  15   0  7520 7520 5764 R 73.8  4.7   0:02.22 postmaster
    4 root       9   0     0    0    0 S  5.3  0.0   0:03.02 kswapd
24609 postgres  11   0  1036 1036  848 R  1.0  0.6   0:00.29 top
24436 postgres   9   0  2992 2992 2924 S  0.3  1.9   0:00.38 postmaster
    1 root       9   0   500  500  448 S  0.0  0.3   0:00.37 init
    2 root       9   0     0    0    0 S  0.0  0.0   0:00.80 keventd
    3 root      19  19     0    0    0 S  0.0  0.0   0:00.02 ksoftirqd_CPU0
    5 root       9   0     0    0    0 S  0.0  0.0   0:00.08 bdflush
```

Figura 17-1: Salida de comando "top"

vmstat

Estadísticas sobre la memoria virtual

```
postgres@debian:~$ vmstat 5
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 0  0      0  19396  11708  86196    0    0     6    16  102   14  0  1 99  0
 0  0      0  19388  11712  86196    0    0     0    10  104   14  0  0 100  0
 0  0      0  19388  11712  86196    0    0     0     0  100   12  0  0 100  0
 0  0      0  19388  11712  86196    0    0     0     0  100   11  0  0 100  0
 0  0      0  19388  11712  86196    0    0     0     0  100   12  0  0 100  0
 0  0      0  19388  11712  86196    0    0     0     0  100   12  0  0 100  0
 0  0      0  19388  11712  86196    0    0     0     0  100   12  0  0 100  0
```

Figura 17-2: Salida de comando “vmstat”

iostat

Estadísticas del uso de los discos

```
$ iostat 5
      tty             sd0             sd1             sd2                % cpu
 tin tout   sps  tps msps   sps  tps msps   sps  tps msps  usr nic sys int idl
   7  119   244   11  6.1     0    0 27.3     0    0 18.1    9   1   4   0  86
   0   86    20    1  1.4     0    0  0.0     0    0  0.0    2   0   2   0  96
   0   82    61    4  3.6     0    0  0.0     0    0  0.0    2   0   2   0  97
   0   65     6    0  0.0     0    0  0.0     0    0  0.0    1   0   2   0  97
  12   90    31    2  5.4     0    0  0.0     0    0  0.0    4   0   3   0  93
  24  173     6    0  4.9     0    0  0.0     0    0  0.0   48   0   3   0  49
   0   91  3594   63  4.6     0    0  0.0     0    0  0.0   11   0   4   0  85
```

Figura 17-3: Salida de comando “iostat”

free

Memoria usada y memoria libre.

```
postgres@debian:~$ free
             total       used       free     shared    buffers     cached
Mem:        159796     140556      19240          0      11748      86264
-/+ buffers/cache:      42544     117252
Swap:       128480          0     128480
```

Figura 17-4: Salida del comand "free"

17.1.2 Uso del Recolector de estadísticas

Si activamos el **recolector de estadísticas** este recoge información de la actividad del servidor. La información que se obtiene tiene que ver con:

- los accesos a las tablas e índices individuales y por páginas,
- instrucciones ejecutadas en los procesos servidores
- lecturas de disco
- uso de la cache
- etc.

Producen una ligera sobrecarga en el servidor, puede ser de 0.5 segundos.

Para ponerlo en marcha hay que modificar el *postgresql.conf:*

- *start_stats_collector = on*
- *stats_command_string, stats_block_level y stats_row_level:* fijan la cantidad de información recopilada

Una vez en marcha, provee información a una serie de tablas o vistas. A continuación vemos algunas de ellas:

pg_stat_activity

Las tablas ***pg_stat_ ** hablan siempre de frecuencias y tipos de accesos.

Información sobre los procesos servidores, una tupla por backend:

```
    Column     |  Type   | Descripcion
---------------+---------+-----------
 datid         | oid     | OID de la base de datos
 datname       | name    | Nombre de la base de datos
 procpid       | integer | PID del backend
 usesysid      | oid     | UID del usuario
 usename       | name    | Nombre de usuario
 current_query | text    | Query actual (solo si superusuario)
```

pg_stat_database

Información sobre actividad en las bases de datos:

```
    Column     |  Type   | Descripcion
---------------+---------+-----------
 datid         | oid     | OID de la BD
 datname       | name    | Nombre de la BD
 numbackends   | integer | No. de backends conectados
 xact_commit   | bigint  | No. de transacciones completas
 xact_rollback | bigint  | No. de transacciones abortadas
 blks_read     | bigint  | No. de bloques leídos
 blks_hit      | bigint  | No. de bloques en cache
```

pg_stat_all_tables

Accesos a las tablas, una tupla por tabla:

```
    Column     |  Type   | Descripcion
---------------+---------+-----------
 relid         | oid     | OID de tabla
 relname       | name    | Nombre de tabla
 seq_scan      | bigint  | Cantidad de seqscans
 seq_tup_read  | bigint  | No. de tuplas devueltas por seqscan
 idx_scan      | numeric | Cantidad de index scans
 idx_tup_fetch | numeric | No. de tuplas devueltas por indexscan
 n_tup_ins     | bigint  | No. de tuplas insertadas
 n_tup_upd     | bigint  | No. de tuplas actualizadas
 n_tup_del     | bigint  | No. de tuplas eliminadas
```

Variantes: *pg_stat_user_tables, pg_stat_sys_tables*

pg_stat_all_indexes

Acceso a los índices

```
    Column     |  Type  | Descripcion
---------------+--------+-----------
 relid         | oid    | OID de la tabla
 indexrelid    | oid    | OID del índice
 relname       | name   | Nombre de la tabla
 indexrelname  | name   | Nombre del índice
 idx_scan      | bigint | Cantidad de scans con este índice
 idx_tup_read  | bigint | Cantidad de tuplas leídas
 idx_tup_fetch | bigint | Cantidad de tuplas devueltas
```

Diferencia entre *tup_read y tup_fetch*: tuplas obsoletas (MVCC)

Variantes: *pg_stat_user_indexes, pg_stat_sys_indexes*

pg_statio_all_tables

Las vistas ***pg_statio_**** hablan de E/S física.

Información sobre E/S en tablas

```
     Column      |  Type   | Descripción
-----------------+---------+-----------
 relid           | oid     | OID de la tabla
 relname         | name    | Nombre de la tabla
 heap_blks_read  | bigint  | No. de bloques leídos de heap
 heap_blks_hit   | bigint  | No. de bloques en cache
 idx_blks_read   | numeric | No. de bloques de índices leídos
 idx_blks_hit    | numeric | No. de bloques de índices en cache
 toast_blks_read | bigint  | No. de bloques leídos de TOAST
 toast_blks_hit  | bigint  | No. de bloques leídos de TOAST en cache
 tidx_blks_read  | bigint  | No. de bloques de índice de TOAST leídos
 tidx_blks_hit   | bigint  | No. de bloques de índice de TOAST en cache
```

Variantes: *pg_statio_user_tables, pg_statio_sys_tables*

A partir de la información obtenida se puede pensar en aumentar los *shared buffers* o cambiar la organización de la tabla.

pg_statio_all_indexes

E/S en índices

```
    Column     |  Type  | Descripcion
---------------+--------+-----------
 relid         | oid    | OID de la tabla
 indexrelid    | oid    | OID del índice
 relname       | name   | Nombre de la tabla
 indexrelname  | name   | Nombre del índice
 idx_blks_read | bigint | Número de bloques leídos
 idx_blks_hit  | bigint | Número de bloques leídos en cache
```

Variantes: *pg_statio_user_indexes, pg_statio_sys_indexes*

pg_statio_all_sequences

E/S en secuencias.

```
  Column   |  Type  | Descripcion
-----------+--------+-----------
 relid     | oid    | OID de la secuencia
 relname   | name   | Nombre de la secuencia
 blks_read | bigint | No. de bloques leídos
 blks_hit  | bigint | No. de bloques leídos en cache
```

Variantes: *pg_statio_user_sequences, pg_statio_sys_sequences*

pg locks

Visualizar los bloqueos que se producen, determinando la tabla con el mayor número de bloqueos no otorgados (causante del cuello de botella),

17.2 Monitorización del uso de los discos:

El DBA no tiene demasiado control, es el propio PostgreSQL quien controla este tema.

Las tablas guardan su información en **una fichero inicial (*heap)*** y luego tienen asociado un **fichero de desborde *(toast)***.

Formas de obtener el uso de disco:

- consultando vistas del catálogo
- usando las extensiones $PGSRC/contrib:
  - *dbsize:* bases de datos y sus tamaños
  - *oid2name:* correspondencia entre nombres de tablas y ficheros

Por ejemplo, consultando vistas del catálogo:

```
-- Tamaño de tabla: consultar la tabla pg class
SELECT  relfilenode, relpages FROM pg_class WHERE relname='tabla';

-- Tamaño de desborde (TOAST):
SELECT relfilenode, relpages FROM pg_class
WHERE relname IN ('pg toast_relfilenode', 'pg_toast_relfilenode_index');

-- Tamaño de índices:
SELECT c2.relname, c2.relpages FROM pg_class c, pg_class c2, pg_index i
WHERE c.relname ='tabla' AND c.oid = i.indrelid
AND c2.oid = i.indexrelid ORDER BY c2.relname;
```

17.3 Funciones para bases de datos

PostgreSQL proporciona una serie de funciones que nos devuelven información de monitorización del sistema.

17.3.1 Sobre transacciones, bloques, y tuplas

Cantidad de backends conectados a la BD

```
pg_stat_get_db_numbackends(oid)
```

Cantidad de transacciones completas (COMMIT)

```
pg_stat_get_db_xact_commit(oid)
```

Cantidad de transacciones abortadas (ROLLBACK)

```
pg_stat_get_db_xact_rollback(oid)
```

Cantidad de bloques leídos

```
pg_stat_get_db_blocks_fetched(oid)
```

Cantidad de bloques leídos encontrados en cache

```
pg_stat_get_db_blocks_hit(oid)

parámetro es OID de la BD
```

Cantidad de scans (index scan o seqscans)

```
    pg_stat_get_numscans(oid)
```

Cantidad de bloques leídos de la tabla o índice

```
    pg_stat_get_blocks_fetched(oid)
```

Cantidad de bloques leídos encontrados en cache

```
    pg_stat_get_blocks_hit(oid)
```

Número de tuplas retornadas por el scan

```
    pg_stat_get_tuples_returned(oid)
```

Número de tuplas leídas por el scan

```
    pg_stat_get_tuples_fetched(oid)

parámetro es OID de tabla o índice
```

Cantidad de tuplas insertadas

```
    pg_stat_get_tuples_inserted(oid)
```

Cantidad de tuplas actualizadas

```
    pg_stat_get_tuples_updated(oid)
```

Cantidad de tuplas eliminadas

```
    pg_stat_get_tuples_deleted(oid)

parámetro es OID de tabla
```

17.3.2 Funciones para backends

Retorna lista de BackendIDs en uso

```
    pg_stat_get_backend_idset()
```

PID del backend

```
    pg_stat_get_backend_pid(integer)
```

OID de base de datos a que está conectado un backend

```
    pg_stat_get_backend_dbid(integer)
```

OID de usuario con que está conectado un backend

```
    pg_stat_get_backend_userid(integer)
```

Consulta en ejecución del backend (NULL si no es superusuario)

```
pg_stat_get_backend_activity(integer)

parámetro es Backend ID
```

17.3.3 Funciones – ejemplo:

```
SELECT pg_stat_get_backend_pid(S.backendid) AS procpid,
       pg_stat_get_backend_activity(S.backendid) AS current_query
FROM (SELECT pg_stat_get_backend_idset() AS backendid) AS S;
```

Capítulo 18. Afinamiento del Servidor

Para saber las operaciones de afinamiento y mejora de rendimiento, lo primero es recordar la arquitectura de PostgreSQL:

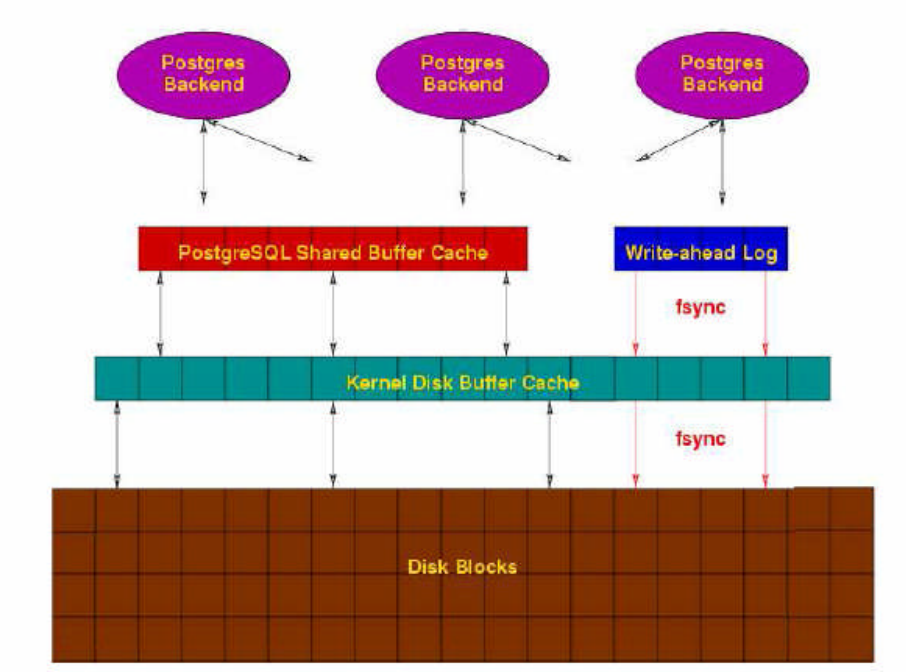

Figura 18-1: Arquitectura de PostgreSQL

Entonces, los elementos sobre los que podemos actúar son:

- Gestión del diario (WAL)
- Buffers de diario (cache WAL)
- Cache de base de datos
- Acceso a los discos

18.1 Gestión del diario (WriteAheadLogging) en pg_xlog

Cualquier escritura en los ficheros de datos (tablas e índices) debe ser posterior a la escritura en el diario, esto permite que no sea necesario escribir a páginas de datos después de cada COMMIT.

Ante un fallo de sistema se puede recuperar el estado correcto a partir del diario.

Beneficios:

- Reduce significativamente el número de escrituras a disco. Sólo es necesario escribir el buffer de diario en el momento de la confirmación y sSe puede aprovechar una escritura para llevar información de varias transacciones.
- Consistencia de las páginas de datos: antes del WAL, PostgreSQL no podía asegurar la consistencia de las páginas en caso de fallo: entradas de índices apuntando a filas inexistentes, pérdidas de entradas de índices en operaciones de división (árbol-B), páginas de tablas o índices totalmente corrompidas por escrituras parciales de las mismas.

Configuración:

Las escrituras en los WAL vienen dadas por los puntos de control, checkpoint, entonces todos los parámetros que manejan los checkpoints influyen el el funcionamiento del WAL:

- *checkpoints_segments* (nº de ficheros WAL) *y checkpoint_timeout* (tiempo que tardan en escribirse las páginas sucias de la cache al disco) marcan cuando se produce un punto de control.
- *checkpoint_warning*

18.2 Buffers de diario (cache WAL)

También influye los *buffers de diario WAL,* los cuales realizan las siguientes operaciones:

- *LogInsert*: un proceso servidor escribe una entrada de una operación
- *LogFlush*: una confirmación provoca la escritura de los buffers al diario en disco

Problemas principales	Solución
Al intentar guardar uno nuevo, no queda espacio en la cache. Escribir algunos buffers a disco: es lento o muy lento (puede ser que sea necesario crear un nuevo fichero). El proceso servidor debe esperarse: las operaciones que necesitan generar la nueva entrada están a la espera.	aumentando el tamaño de la cache de WAL, incrementando el número de buffers WAL (WAL_BUFFERS=n°)
Sistemas muy sobrecargados, no escribe y libera espacio en los buffers WAL lo suficientemente rápido.	
Que la escritura de los buffers de diario a disco no sea real, al usarse los buffers intermedios del sistema	Utilizar el método que usa PostgreSQL para forzar al SO que escriba a disco los buffers (parámetros FSYNC=true y WAL_SYNC_METHOD=método)
Eficiencia: cada COMMIT requiere una escritura de diario	Consiguiendo que el sistema pueda llevarse información de varias transacciones en una sola escritura de diario, para ello: o COMMIT DELAY: cuantos microsegundos el sistema debe esperar después de recibir el COMMIT para realizar la escritura de las entradas del diario. (Permite que se escriba entradas de otras transacciones en la misma operación). Por debajo de 10000 no está claro que tenga efecto.

	o COMMIT_SIBLING: cuántas sesiones con transacciones activas como mínimo han de existir para que el servidor retarde la-escritura de buffers en diario. o Si FSYNC=false O sesiones traes. act.<COMMIT_SIBLING, entonces no hay retardo de COMMIT DELAY

18.3 Cache de base de datos

Se trata aquí de **minimizar el acceso al disco**- Cualquier instrucción primero busca los datos en la cache, si no están los trae de los discos y trabaja en la cache con ellos.

Se ubica en la memoria compartida del ordenador:

```
# /sbin/sysctl -A I grep shm
```

kernel.shmmni = 4096
kernel.shmall = 2097152
kernel.shmmax = 33554432

A veces es necesario incrementarla

Postgres ocupa de SHMMAX:

```
250 kB + 8.2 kB * shared buffers + 14.2 kB * max connections
```

Podemos usar las vistas de rendimiento *pg_statio_** que indican para cada tabla, índice y secuencia los bloques leídos de disco y de la cache. Permite analizar y cambiar el tamaño o la forma de consultas.

El problema habitual aquí es que la cache no es muy efectiva y se detecta porque el porcentaje de bloques encontrados en la cache es bajo. La causa es el tamaño de cache insuficiente.

Aumentarlo modificando el parámetro SHARED_BUFFERS.

Otro problema es que hay **consultas con rastreo secuencial sobre tablas muy grandes** que no caben en la cache e impiden su utilización, modificar estas consultas e introducir índices para que el sistema cambie a rastreo indexado.

18.4 Rendimiento del acceso a los discos:

Evitar la competición por su uso: separar operaciones diferentes en discos diferentes: las operaciones de manipulación (sobre ficheros de datos y ficheros de índice) en un disco y la gestión de diarios en otro.

Realizar las tareas de mantenimiento rutinarias: REINDEX, VACUUM o mejor AUTOVACUUM

18.4.1 Organización de los datos

La forma en la que organicemos el cluster también influye en el rendimiento. Nos interesa más aquí la organización a nivel físico o lógico-físico.

A partir de la versión 8.0, existen los *TABLESPACES* que nos permiten dividir físicamente la base de datos en tablespaces, los cuales no tienen porque estar en los mismos discos ni, por supuesto, en un directorio dentro del cluster.

```
$ mkdir mnt/sdal/postgresql/data
$ psql
postgres#= CREATE TABLESPACE eldiscorapido LOCATION
     '/mnt/sdal/postgresql/data';
```

así podemos crear varias localizaciones diferentes para los datos, aunque parezca que estamos dentro del mismo cluster, porque el directorio *pg_tblspc* contiene enlaces simbólicos a estos directorios.

Además. hay que tener clara la estructura física de las tablas e índices, para ello hay que tener claro que la entrada y salida se realiza mediante ***paginación***, siendo cada página de 8K. Así, del número de páginas de la tabla/índice se deriva el tamaño de la tabla y del nº de filas/entradas, el ratio de filas/entradas por página. Entonces, debido a MVCC, una modificación en una fila, crea otra fila, pero el nº de filas sigue siendo el mismo, lo que aumenta realmente es el nº de páginas. Esto se arregla cuando se hace VACUUM FULL.

Si una fila no cabe en una página, , PostgreSQL la divide y escribe una parte en una tabla ***TOAST*** *(The oversized attribute storage technique).* Esta tabla actúa como una extensión, manteniéndose los valores que son demasiado grandes para que estén en línea. Esta misma técnica se aplica a los índices.

Hay una tabla TOAST para cada tabla que tenga alguna fila que no quepa: *pg_toast_nnnnn,* donde nnnnn es el OID de la tabla.

Para ver los tamaños que ocupan nuestros objetos:

```
-- tamaño de una tabla

SELECT relfilenode, relpages * 8 AS kilobytes FROM pg_class
WHERE relname = 'clientes';

      relfilenode       kilobytes
         16806             480
Después podemos mirar lo que ocupa en el disco el fichero 16806

-- tamaño del toast:

SELECT relname, relpages * 8 AS kilobytes FROM pg_class
WHERE relname = 'pg_toast_16806' OR relname = 'pg_toast_16806_index'
ORDER BY relname;

-- tamaño del índice:

SELECT c2.relname, c2.relpages * 8 AS kilobytes FROM pg_class c, pg_class c2,
pg_index i WHERE c.relname = 'clientes' AND
c.oid=i.indrelid AND c2.oid=i.indexrelid ORDER BY c2.relname;

-- ver qué tablas son más grandes
```

```
SELECT relname, relpages * 8 AS kilobytes FROM pg_class
ORDER BY relpages DESC
```

Capítulo 19. Afinamiento: Optimización de Consultas

En el capítulo anterior hemos visto cuestiones para aumentar el rendimiento de la base de datos desde un punto de vista del hardware (memoria, disco, ficheros, etc.), en este capítulo entramos más en cuestiones de mejora desde "dentro" de la base de datos, bien dando pautas para una mejor definición de los objetos o bien pautas que sirvan para escribir consultas más eficientes.

19.1 Rendimiento de los índices

Se trata de agilizar el acceso a los datos,

- en las **búsquedas**: Cuando en la condición de la instrucción interviene alguna columna que es utilizada en algún índice, el sistema agiliza la búsqueda de la información cambiando a rastreo indexado sobre este índice.

- en las **ordenaciones**: Cuando la instrucción exige ordenar la información, el sistema comprueba si existe algún índice que le permite devolver la información ya ordenada.

Entonces, en las búsquedas, conforme la condición tenga más columnas de la clave del índice y su uso coincida con su orden, su utilización será más eficiente, es decir, las mejoras consisten en modificar la condición y/o crear nuevos índices.

Siempre, con cuidado con la sobrecarga, porque las modificaciones serán más lentas.

Para las ordenaciones: si el orden físico de almacenamiento de las filas de la tabla coincide con el orden del índice, el rendimiento sube mucho debido al mayor éxito de localizaciones en la cache, así las soluciones podrían ser:

- recrear la tabla con ese orden usando "*CREATE TABLE... AS SELECT ... ORDER BY"*, "*DROP TABLE", "ALTER TABLE" ... "RENAME"*
- crear un cluster de tabla con "*CLUSTER [[índice ON] tabla]"* que reordena físicamente las filas de la tabla según el orden del índice (bloqueo exclusivo en lectura y escritura) y el sistema recuerda la forma de agrupar la tabla

19.1.1 Tipos de índices

El definir índices acelerará el rendimiento de las búsquedas siempre que la consulta tenga condiciones asociadas a los índices. Existen varios tipos de índice:

- ***árbol-B (BTREE)***: valor por defecto, admiten varios predicados (>, <, =, LIKE, etc), multicolumna y unicidad.
- ***árbol-R (RTREE)***: indicado para datos espaciales.
- ***Hash***: sólo admite "=", mal rendimiento

- ***GiST***: índice btree que puede ser extendido mediante la definición de nuevos predicados de consulta, también es multicolumna.

```
// un btree
CREATE INDEX mitabla_idx ON tabla (edad);

// un HASH
CREATE INDEX mitabla_idx ON tabla USING HASH (edad);
```

19.2 Ejecución de consultas

En la ejecución de consultas participan tres elementos:

Analizador:

- Análisis sintáctico y semántico
- Generación del árbol de análisis

Planificador:

- Generación de planes de ejecución
- Plan de operadores: rastreos secuenciales, rastreos indexados,métodos de concatenación de tablas y ordenaciones

Optimizador:

- Búsqueda del plan más económico
- Estimación del coste de cada operador
- Se puede influir en los costes asignados a cada operador

Mediante ***EXPLAIN*** podemos ver el plan de ejecución de una consulta, y ver si se puede rediseñar de un modo mejor, al ejecutar *explain* sobre una consulta, nos permitirá:

- ver el plan de ejecución de consulta
- ver si se están usando índices
- usa estadísticas para generar plan óptimo
- ejecutando comandos *"SET enable_* TO off"* nos permite averiguar que haría en otro caso
- EXPLAIN ANALYZE muestra costo real de ejecución

Veamos los distintos tipos de rastreo que puede usar PostgreSQL:

19.2.1 Rastreo secuencial (SEQ SCAN):

Revisa la tabla de principio a fin, Evalúa la condición de la consulta para decidir si incorporar la fila al resultado. Se pasa por encima de las filas muertas y rl coste inicial es siempre O (recuperación inmediata de la primera fila)

19.2.2 Rastreo indexado (INDEX SCAN):

Utiliza un índice de la tabla implicada, no tiene que revisar todas las filas de la tabla, Pero algunos bloques (páginas) pueden ir y venir varias veces, si la tabla está muy desordenada según el índice (no se ha hecho un CLUSTER recientemente), el resultado es ordenado según el orden del índice.

Sólo se utiliza en índices árbol-B, árbol-R y GiST.

El planificador/optimizador utiliza este rastreo cuando puede reducir el número de filas a revisar o la necesidad de ordenar el resultado.

19.2.3 Ordenación (SORT):

La ordenación es de dos tipos según donde se haga:

1. en memoria principal: si cabe en la memoria de ordenación (tamaño determinado por el parámetro work_mem)
2. en disco: se particiona en varias ordenaciones y luego se fusionan los resultados parciales

No reduce el tamaño del resultado, procesa toda la información antes de devolver ninguna fila

19.2.4 Unicidad (UNIQUE):

- Eliminación de duplicados
- Requiere la ordenación del conjunto de entrada
- Puede devolver la primera fila antes de terminar de procesar

19.2.5 Operadores LIMIT.y OFFSET (LIMIT):

```
SELECT ... OFFSET x LIMIT y;
```

Pasa por alto las primera x filas del conjunto, devuelve las y siguientes y tampoco tiene en cuenta las siguientes

19.2.6 Agregado (AGGREGATE):

Usa este operador cuando la consulta incluye alguna función agregada, p.ej. AVG(..) Revisa todas las filas del conjunto y calcula el valor agregado (una única fila)

■

19.2.7 Añadir (APPEND):

Usa este operador para implementar UNION.

Devuelve todas las filas del primer conjunto, luego las del segundo, y así hasta las del último conjunto. Su coste es la suma de los costes de obtener los conjuntos orígenes.

Interrelación con la herencia. También se usa ahí.

19.2.8 Resultado (RESULT):

Para obtención de cálculos intermedios no relacionados con la consulta, evalúa partes o condiciones constantes llamadas a funciones numéricas (no agregadas).

19.2.9 Bucle anidado (NESTED LOOP):

Para la realización de la concatenación entre tablas Para cada fila de la tabla "externa" busca las filas de la tabla interna que cumplan la condición de concatenación

¡Ojo! Sólo se usa para los joins internos

19.2.10 Concatenación por fusión (MERGE JOIN):

Para la realización de la concatenación entre tablas. Los dos conjuntos deben estar ordenados por la columna de la concatenación. Utiliza el hecho de estar ordenados los conjuntos para realizar la concatenación. Se usa para todo tipo de joins y para la unión

19.2.11 Hash y concatenación hash (HASH y HASH JOIN):

Se usan combinadamente parada realización de la concatenación entre tablas. La concatenación hash empieza utilizando el operador hash sobre la tabla interna que crear un índice hash temporal sobre esta tabla. Luego la concatenación comienza tomando una fila de la tabla externa y buscando, haciendo uso del índice hash, alguna fila de la tabla interna que combine Se usa para joins internos, joins por la izquierda y para la unión

19.2.12 Agrupar (GROUP).

Se usa para realizar la agrupación en las consultas, dependiendo de si se va a realizar cálculos agregados, se añade una fila con nulos a los grupos formados como separador.

19.2.13 Rastreo de subconsulta y subplan (SUBQUERY SCAN y SUBPLAN):

Se usan para subconsultas:

- El rastreo de subconsulta se usa para cada parte del UNION
- Subplan se usa para las subconsultas generales.

Se usan para operaciones internas y generalmente se pueden omitir

19.2.14 Rastreo por TID (identificador de tupla) (TID SCAN):

Se usa con poca frecuencia

TID (corresponde a la pseudo columna ***ctid***) está compuesto del bloque donde se encuentra la tupla y el número de tupla dentro del bloque. Sólo se puede utilizar dentro de una transacción. Es la forma más rápida de localizar una tupla

19.2.15 Materializar (MATERIALIZE):

Se usa con algunas operaciones de subconsultas

El planificador/optimizador puede decidir que es más económico materializar la subconsulta, que repetirla varias veces (suele pasar en las concatenación por fusión)

19.2.16 Operadores conjuntistas (SETOP):

Intersección, Intersección total, diferencia y diferencia total

Requieren dos conjuntos, que combinan en una lista ordenada y luego identifican grupos de filas idénticas (se contabilizan cuantas y de qué conjunto provienen). Posteriormente, dependiendo del operador se termina cuantas filas pasan al resultado final
Se usan para INTERSECT, INTERSECT ALL, EXCEPT y EXCEPT ALL

Capítulo 20. Postgresql – Otros aspectos

Nota: Al final del manual, en el Anexo C, hay más detalles sobre algunos de estos puntos.

20.1 OIDs y otras pseudo-columnas del sistema

En PostgreSQL todos los objetos están identificados por el **OID**, que es un número de 32 bits, hasta 4.294.967.295 siempre distintos, pero que si se agota, hay que repetir.

Todos los objetos, tablas, índices, triggers, hasta las filas tienen un OID único global a todo el cluster.

Los OIDs son permanentes duran toda la vida del cluster, pero no si se copian o mueven los datos. Las tablas tienen por defecto varias columnas de sistema adicionales:

- ***oid:*** *oid de una fila, nº secuencial automáticamente añadido a todas las filas de las tablas*
- ***tableoid:*** oid de la tabla que contiene la fila
- ***xmin:*** XID de la transacción que insertó la fila
- ***cmin:*** el identificador de instrucción (0, 1, …) dentro de la transacción que insertó la fila.
- ***xmax:*** XID de la transacción que borró la fila (vale 0 para las versiones no borradas)
- ***cmax:*** el identificador de instrucción que borró la fila
- ***ctid:*** posición física de la versión de la fila en la tabla, puede ser usado para localizar la versión de la fila rápidamente, cambia cada vez que se hace vacuum.

En el CREATE TABLE existe la opción WITHOUT OIDS, que en la versión 8 es la opción por defecto.

En una consulta no salen los OIDs, salvo que lo digamos explícitamente:

```
SELECT oid FROM tabla;
```

Los **OIDs no sirven como claves primarias**, no son modificable, mediante INSERT no pueden insertar OIDs repetidos (pero existen mecanismos para hacerlo).

Existe también el **tipo OID** que permite definir columnas de ese tipo, se suelen usar para hacer referencias (aunque no es recomendable) o para el tratamiento de campos LOB.

20.2 Secuencias

Son contadores que se crean con:

```
CREATE SEQUENCE nombreseq;
```

se manipulan con:

- `nextval()`: retorna un valor y lo incrementa
- `currval()`: retorna un valor
- `setval()`: establece un valor

Se suelen usar en campos de clave primaria secuenciales:

```
-- de este modo, creamos la tabla haciendo uso de una secuencia que ya existe:

CREATE TABLE cliente (
    ident INTEGER DEFAULT nextval('nombre_secuencia')
    [,otros_campos]);

o bien, más cómodo, pero crea la tabla la secuencia (con nombre aleatorio):

CREATE TABLE cliente (
ident serial,
…
);
```

Claves Primarias: ¿OIDs, Secuencias, Atributos?

Dado que tenemos los OIDs y los campos SERIAL, nos planteamos **la estrategia para definir la mejor clave primaria** para una tabla. Hay tres opciones para identificar filas:

- Si el atributo tradicional es representativo y siempre existente, p. ej, DNI, elegirlo como CP es una buena opción.
- Usar una secuencia como CP es muy buena opción.
- Y usar un OID es muy mala opción.

¿Y para hacer referencias?:

- No usar el OID. Usar valores y claves ajenas tradicionales a la CP de la otra relación (sea secuencia o no) o a otro atributo con unicidad.
- Usar el OID. MALA OPCIÓN. LA BD NO SERÁ PORTABLE!!

Los OIDs se suelen usar para incorporar **objetos grandes BLOBs**.

20.3 Tipos de datos Básicos en PostgreSQL

- Lógicos: BOOLEAN
- Carácter y cadenas: CHAR(.), VARCHAR(.), TEXT)
- Numéricos exactos: NUMERIC(.,.), INTEGER, INT2, INT4, INTB; OID, SMALLINT
- Numéricos aproximados: FLOAT, FLOAT4
- Tiempo: DATE, TIME, TIMESTAMP
- Intervalo: INTERVAL
- Geométrico. POINT, LSEG, PATH, BOX, CIRCLE, POLYGON
- Redes: INET, CIDR, MACADDR

20.4 Tipos compuestos en PostgreSQL

- Vectores (tablas)
- Geométricos
- Red
- BLOBs
- Definidos por el usuario

20.5 Otras Particularidades del Lenguaje SQL:

20.5.1 Comando COPY:

```
COPY tabla TO 'fichero';
COPY tabla FROM 'fichero';
```

20.5.2 Réplica de tablas:

```
SELECT … INTO …
```

20.5.3 Tablas temporales

```
CREATE TEMPORARY TABLE tmptbl
```

20.5.4 Herencia en Tablas

,La herencia es a nivel de datos, no de la estructura de las tablas.

- la tabla hija puede tener más atributos que la madre
- todas las filas insertadas en la tabla hija, pasan a estar en la tabla madre, pero no viceversa
- los atributos heredados modificados en la tabla hija se modifican en la fila correspondiente de la tabla madre

- el OID de las filas comunes madre-hija es el mismo
- se permite herencia múltiple
- los procedimientos y disparadores no se heredan
- las restricciones solo afectan a las tuplas propias de la relación, no a las que se derivan de la generalización

```
CREATE TABLE mare (col1 INTEGER);
CREATE TABLE filla (col2 INTEGER) INHERITS (mare));

-- para mostrar solo los datos de la tabla madre:
SELECT * FROM ONLY MARE;
```

20.5.5 Extensiones en consultas

- SELECT * FROM funcion();
- FROM tabla1 JOIN tabla2
- SELECT CASE
- LIMIT Y OFFSET

Apéndice A. Comando psql

La herramienta canónica para trabajar en modo línea de comandos con PostgreSQL es psql. Aquí tenemos una herramienta completa para poder manipular las bases de datos.

Modo de empleo:

```
psql [opciones] [base_datos]
```

Proporciona numerosos meta-comandos y características tipo shell que facilitan la construcción de scripts y la automatización de tareas. Acepta opciones en la línea de comandos al momento de ejecución. Veamos las principales opciones:

```
[base_datos]      si se omite se conecta a una base de datos llamada "postgres",
se puede llamar también con el parámetro “-d”
Opciones generales:
  -d BASE-DE-DATOS nombre de base de datos a conectarse (por omisión: "postgres")
  -c COMANDO ejecutar sólo un comando (SQL o interno) y salir
  -f ARCHIVO ejecutar comandos desde archivo, luego salir
  -l listar bases de datos, luego salir
  -v NOMBRE=VALOR definir variable NOMBRE de psql a VALOR
  -X no leer archivo de configuración (~/.psqlrc)
  --help mostrar una ayuda, luego salir
  --version mostrar información de versión, luego salir
  man: muestra la ayuda del man

Opciones de entrada y salida:
  -a               mostrar los comandos del script
  -e               mostrar comandos enviados al servidor
  -E               mostrar consultas generadas por comandos internos
  -q               modo silencioso (sin mensajes, sólo resultado de consultas)
  -o ARCHIVO       enviar resultados de consultas a archivo (o |comando)
  -n               deshabilitar edición de línea de comando (readline)
  -s               modo paso a paso (confirmar cada consulta)
  -S               modo de líneas (fin de línea termina el comando SQL)
  -L ARCHIVO     manda el log de la sesión a un archivo

Opciones de formato de salida:
  -A               modo de salida desalineado
  -H               modo de salida en tablas HTML
                   (-P format=html)
  -t               mostrar sólo filas (-P tuples_only)
  -T TEXTO         definir atributos de marcas de tabla HTML (ancho, borde)
                   (-P tableattr=)
  -x               activar modo expandido de salida de tablas (-P expanded)
  -P VAR[=ARG]     definir opción de impresión VAR en ARG (ver comando \pset)
  -F CADENA        definir separador de columnas (por omisión: «|»)
                   (-P fieldsep=)
  -R CADENA        definir separador de filas (por omisión: salto de línea)
                   (-P recordsep=)

Opciones de conexión:
  -h ANFITRIÓN     nombre del anfitrión o directorio de socket
                   (por omisión: «socket local»)
  -p PORT          puerto del servidor de bases de datos (por omisión: "5432")
  -U NOMBRE        nombre de usuario de la base de datos (por omisión:
                   se conecta con el mismo usuario que en el SO)
  -W               preguntar contraseña (debería suceder automáticamente)
```

Desde este comando es sencillo conectarse a cualquier base de datos en cualquier máquina, siempre que el sistema nos lo permita. Se pueden ejecutar comandos leídos desde archivos, utilizar la redirección de ficheros de entrada y salida, así como usarlo dentro de scripts del SO.

Entre las opciones más útiles, están `-h servidor` (conexión al host servidor) y `-d base` (conexión a la base de datos base), en cuyo caso también podemos indicarle la opción `-u` para que nos solicite el usuario y la contraseña, en caso de que nos conectemos como un usuario distinto al de la sesión actual.

Otra opción muy útil es `-l` para listar todas las bases existentes:

Ejemplo A.1. listar todas las bases de datos

```
psql -l -U postgres
      Listado de base de datos
  Nombre   |  Dueño   | Codificación
-----------+----------+--------------
 postgres  | postgres | LATIN1
 template0 | postgres | LATIN1
 template1 | postgres | LATIN1
(3 filas)
```

Ejemplo A.2. ejecutar un comando y salir, pasando la clave desde fichero

```
psql -U usu2 -C "update tabla set campo = campo+3" < clave.txt
```

Ejemplo A.3. Hacer que la salida de un comando SQL sea entrada de un comando shell

```
#!/bin/sh

/usr/local/pgsql/bin/psql template1 postgres -A -t -c "SELECT datname
FROM pg_database WHERE datallowconn AND encoding=6 AND NOT datname ~
'.*_s'" | while read D ; do

  echo "Converting $D"
  echo "-- start" > $D.dump
  echo "ALTER DATABASE $D RENAME TO ${D}_s;" >> $D.dump
  echo "CREATE DATABASE $D ENCODING='LATIN1' TEMPLATE=template0;" >> $D.dump
  echo "\\connect $D" >> $D.dump
  /usr/local/pgsql/bin/pg_dump -U postgres $D >> $D.dump
  /usr/local/pgsql/bin/psql -v ON_ERROR_STOP=1 -U postgres template1 -f $D.dump
  echo Exitcode $?
done
```

Nota: si se quiere disponer de edición en la línea de comandos es necesario tener instalada la libreria `readline`.

Para más detalles ver el manual de referencia [http://www.postgresql.org/docs/8.1/static/app-psql.html].

Metacomandos

Cuando ejecutamos **psql**, además de poder escribir sentencias SQL, podemos hacer uso de los metacomandos que incluye este programa. Al iniciar el programa, sale por pantalla:

```
    \h para ayuda de comandos SQL
    \? para ayuda de metacomandos psql
    \g o or termine con punto y coma para ejecutar una consulta
    \q para salir
```

Pulsando \? sale la ayuda de los metacomandos:

```
General
  \c[onnect] [BASE-DE-DATOS|- [USUARIO]]
                 conectar a una nueva base de datos (actual: «postgres»)
  \cd [DIR]      cambiar el directorio de trabajo
  \copyright     mostrar términos de uso y distribución de PostgreSQL
  \encoding [CODIFICACIÓN]
                 mostrar o definir codificación del cliente
  \h [NOMBRE]    mostrar ayuda de sintaxis de comandos SQL,
                 * para todos los comandos
  \q             salir de psql
  \set [NOMBRE [VALOR]]
                 definir variables internas,
                 listar todas si no se dan parámetros
  \timing        mostrar tiempo de ejecución de comandos
                 (actualmente desactivado)
  \unset NOMBRE  indefinir (eliminar) variable interna
  \! [COMANDO]   ejecutar comando en intérprete de comandos,
                 o iniciar intérprete interactivo

Búfer de consulta
  \e [ARCHIVO]   editar el búfer de consulta (o archivo) con editor externo
  \g [ARCHIVO]   enviar búfer de consulta al servidor
                 (y resultados a archivo o |comando)
  \p             mostrar el contenido del búfer de consulta
  \r             reiniciar (limpiar) el búfer de consulta
  \w ARCHIVO     escribir búfer de consulta a archivo

Entrada/Salida
  \echo [CADENA] escribir cadena a salida estándar
  \i ARCHIVO     ejecutar comandos desde archivo
  \o [ARCHIVO]   envíar resultados de consultas a archivo o |comando
  \qecho [CADENA] escribir cadena a salida de consultas (ver \o)

Informacional
  \d [NOMBRE]    describir tabla, índice, secuencia o vista
  \d{t|i|s|v|S} [PATRÓN] («+» para obtener más detalles)
                 listar tablas/índices/secuencias/vistas/tablas de sistema
  \da [PATRÓN]   listar funciones de agregación
  \db [PATRÓN]   listar tablespaces («+» para más detalles)
  \dc [PATRÓN]   listar conversiones
  \dC            listar conversiones de tipo (casts)
  \dd [PATRÓN]   listar comentarios de objetos
  \dD [PATRÓN]   listar dominios
  \df [PATRÓN]   listar funciones («+» para más detalles)
  \dg [PATRÓN]   listar grupos
```

```
  \dn [PATRÓN]    listar esquemas («+» para más detalles)
  \do [NOMBRE]    listar operadores
  \dl             listar objetos grandes, lo mismo que \lo_list
  \dp [PATRÓN]    listar privilegios de acceso a tablas, vistas y secuencias
  \dT [PATRÓN]    listar tipos de dato («+» para más detalles)
  \du [PATRÓN]    listar usuarios
  \l              listar todas las bases de datos («+» para más detalles)
  \z [PATRÓN]     listar privilegios de acceso a tablas, vistas y secuencias
                   (lo mismo que \dp)


Formato
  \a              cambiar entre modo de salida alineado y sin alinear
  \C [CADENA]     definir título de tabla, o indefinir si es vacío
  \f [CADENA]     mostrar o definir separador de campos para
                  modo de salida sin alinear
  \H              cambiar modo de salida HTML (actualmente desactivado)
  \pset NOMBRE [VALOR]
                 define opción de salida de tabla
                 (NOMBRE := {format|border|expanded|fieldsep|footer|null|
                 numericlocale|recordsep|tuples_only|title|tableattr|pager})
  \t              mostrar sólo filas (actualmente desactivado)
  \T [CADENA]     definir atributos HTML de <table>, o indefinir si es vacío
  \x              cambiar modo expandido (actualmente desactivado)

Copy, Objetos Grandes
  \copy ...       ejecutar comando SQL COPY con flujo de datos al cliente
  \lo_export LOBOID ARCHIVO
  \lo_import ARCHIVO [COMENTARIO]
  \lo_unlink LOBOID
  \lo_list        operaciones con objetos grandes
```

Apéndice B. Programa pgAdmin III

pgAdmin III es un interfaz de administración de la base de datos PostgreSQL. pgAdmin III puede conectarse a cualquier base de datos PostgreSQL 7.X/8.X usando la librería empotrada nativa `libpq`.

Cuando se configura el lado del servidor, es posible conectarse usando claves cifradas o autenticación SSL. La interfaz de usuario de pgAdmin III está traducida en más de 20 idiomas. Realiza acceso nativo a PostgreSQL (no se necesita la capa ODBC).

El programa está escrito en C++ y usa la excelente herramienta multi plataforma wxWindows, lo cual puede suponer un problema en algunos entornos como Linux, donde habrá dificultades para instalar esta herramienta (por ejemplo, la versión 1.4 de pgAdmin III no se puede instalar en SUSE 9.0).

En todos los entornos, pgAdmin III es una aplicación nativa. La aplicación se ejecuta como código binario, no en una máquina virtual, ofreciendo por tanto un rendimiento muy bueno.

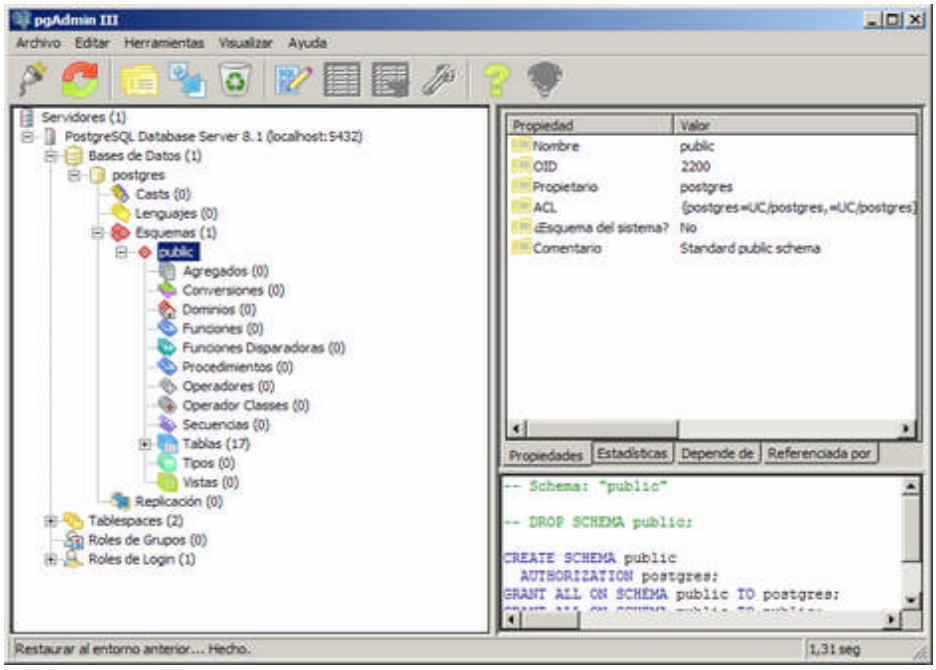

Figura B-1: Pantalla principal de pgAdmin III

Se resumen aquí algunas de las principales funcionalidades de este programa.

1. Conexión con una base de datos

 (Archivo -> Añadir Servidor...)

Lo primero que hay que hacer es conectar con una base de datos en un servidor PostgreSQL, los datos que hay que introducir son los siguientes:

- *direccion*: IP o nombre del host al que conectamos
- *descripción*: descripción de la conexión definida
- *servicio*: nombre del servicio que espera conexiones en un servidor. Normalmente no se pone.
- *puerto*: puerto de conexión a la base de datos
- *SSL*: opciones sobre conexión usando SSL
- *BD mantenimiento*: dentro de una conexión a un servidor, podemos ver cualquiera de las bases de datos, pero hay una que se abre como base de datos de mantenimiento.
- *usuario*: usuario con el que conectamos
- *contraseña*: clave de acceso para el usuario

2. Navegar por los objetos

Con esta herramienta gráfica, una vez establecida una conexión con un servidor de bases de datos, podemos ver todos los objetos definidos, navegando de modo jerárquico, pudiendo ver las propiedadesde cada objeto, que, en función del tipo de objeto, se podrá ver además su código en SQL, o sus propiedades en una ventana, o sus datos, etc.

3. Crear, modificar y borrar objetos

La herramienta posee asistentes en los cuales, de modo gráfico podemos crear objetos del mismo tipo del que tenemos seleccionado, ver y modificar sus propiedades e incluso borrarlos. Dentro de estos asistentes, podemos ver la correspondencia con la sentencia SQL equivalente.

Por ejemplo, si vemos las propiedades de una tabla:

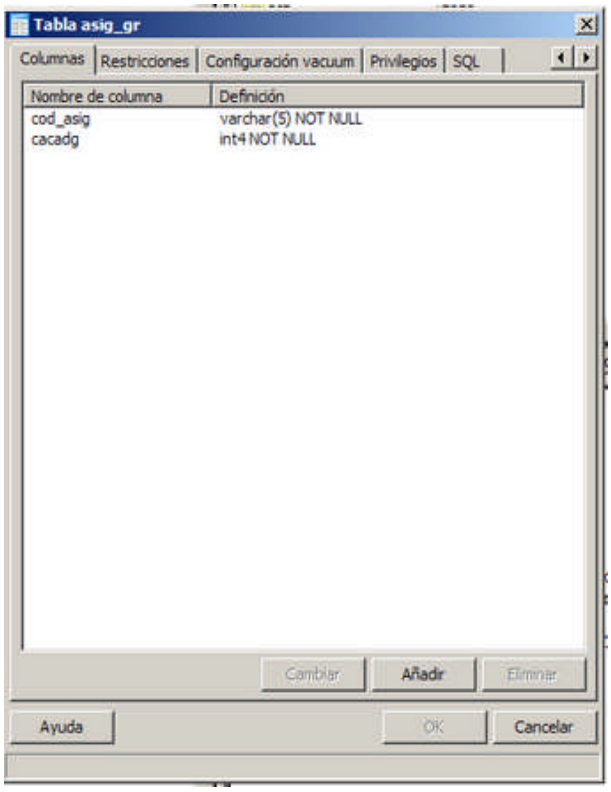

Figura B-2: Propiedades de una tabla

vemos las columnas que tiene definidas, si le damos al botón Añadir, tendremos la posibilidad de añadir nuevas columnas:

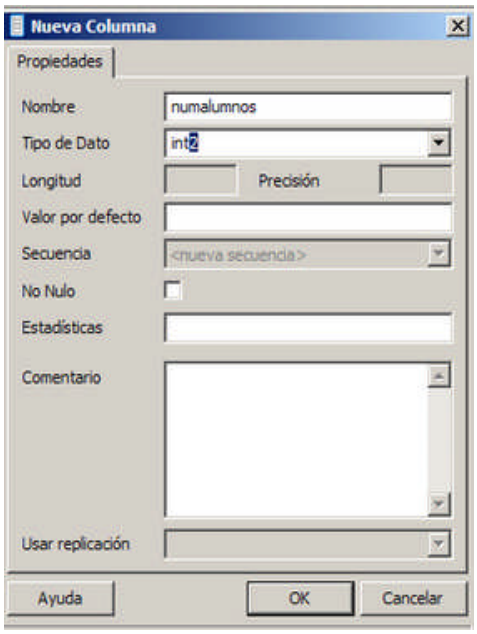

Figura B-3: Añadir nuevas columnas

y cuando terminemos de añadir columnas, si vamos a la pestaña SQL, antes de aceptar los cambios, podremos ver el equivalente en lenguaje SQL:

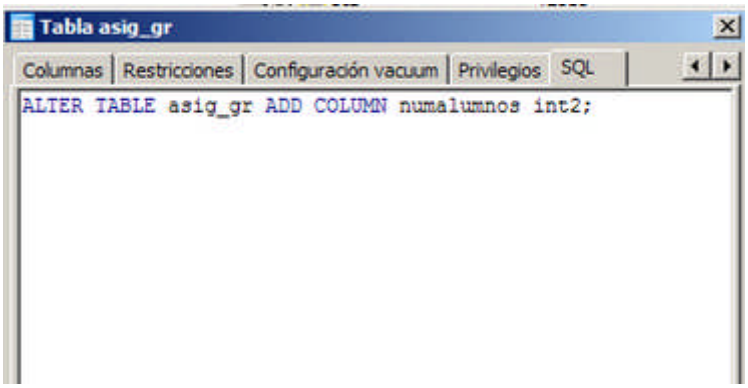

Figura B-4: Pestaña SQL de pgAdminIII

4. Mantener las bases de datos y sus objetos

El programa nos permite mantenimiento de las bases de datos o de las tablas que hay en una de ellas. Por ejemplo, sI vemos el menú contextual de una tabla:

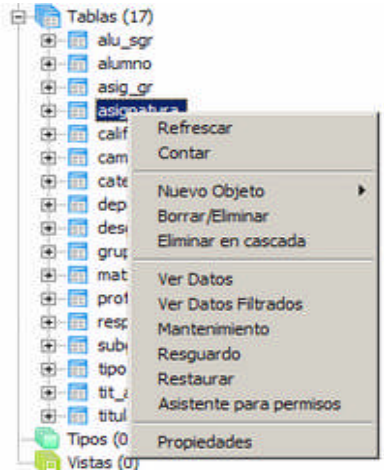

Figura B-5: . Opciones de mantenimiento de una tabla

podemos entrar en las distintas opciones, como:

- **Mantenimiento**

  Se pueden realizar opciones como **vacuum, analyze o reindex**.

Figura B-6: Mantenimiento de una tabla

- **Resguardo** (copia de seguridad o exportar datos)

Realizar copias de seguridad o exportar datos, en formatos propios de PostgreSQL o en formato plano, con distintas opciones.

Figura B-7: Copia de seguridad

- **Restaurar**

Restaurar, recuperar o importar datos de una tabla, en distintos formatos:

Figura B-8: Restaurar datos

5. Ver y filtrar datos

Esta opción nos permite ver los datos de una tabla o consulta. En caso de tener permisos, podemos realizar cambios, inserciones y borrados fácilemente:

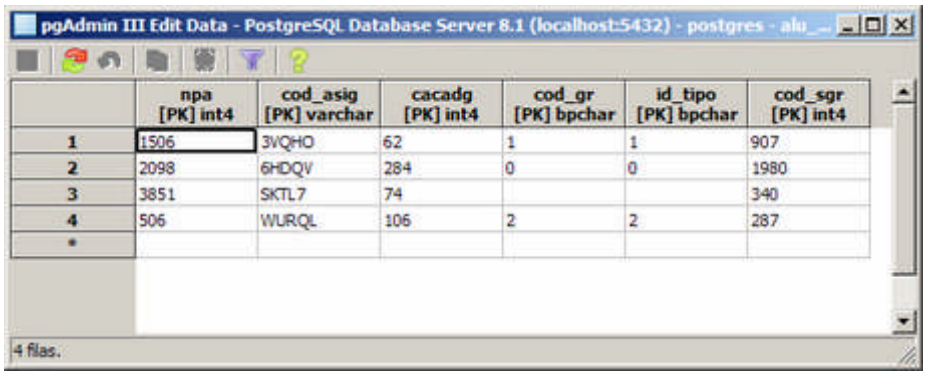

Figura B-9: Ver datos y modificar

6. SQL Interactivo

Desde esta opción disponemos de un potente editor SQL, en el cual podemos ejecutar sentencias SQL, cargar o guardar scripts de comandos, así como se pueden exportar los datos de la salida a un fichero.

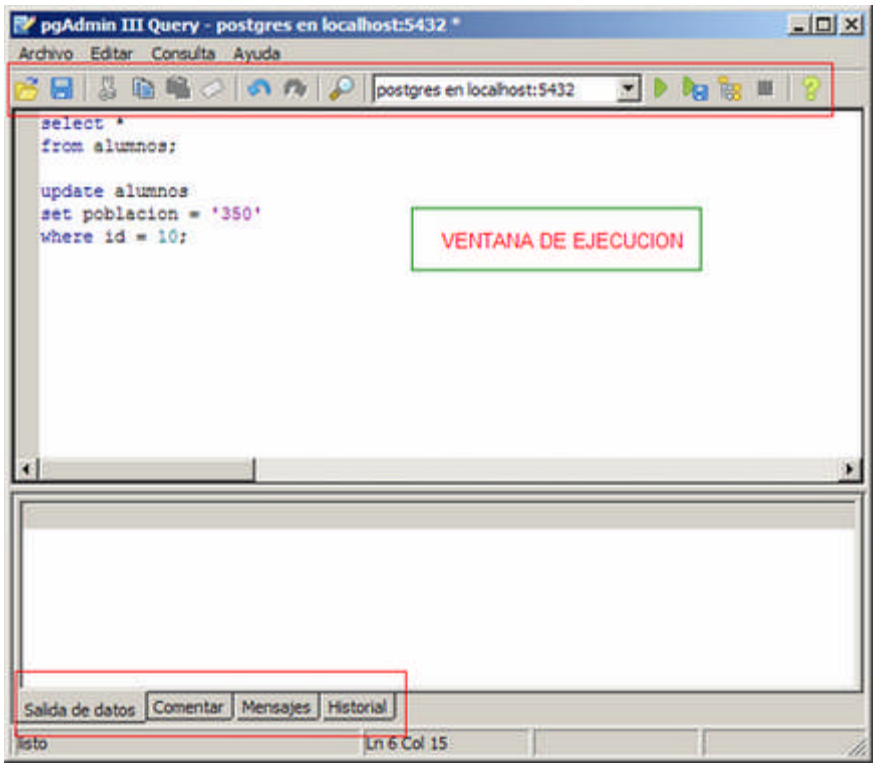

Figura B-10: SQL Interactivo

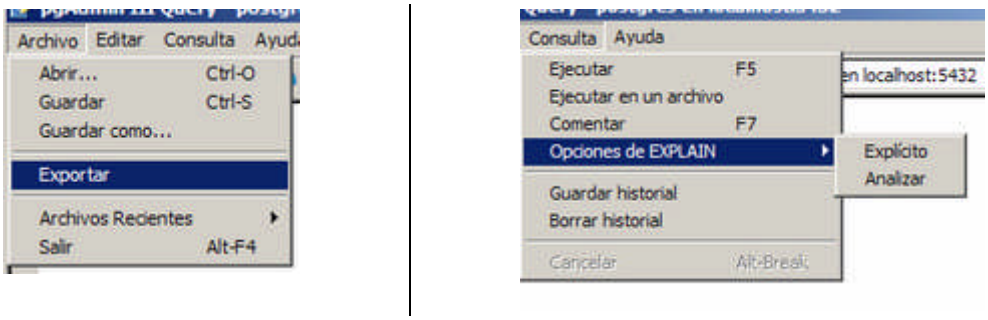

Figura B-11: Consultas SQL: exportar datos y explain

7. Sugerencias de mantenimiento

Cuando este botón está activo sobre un objeto (base de datos o tabla), es porque el programa nos propone sugerencias de mantenimiento sobre dicho objeto (vacuum, reindex, etc.).

8. Opciones del programa

Desde el menú principal se pueden cambiar opciones por defecto del programa, como el idioma en que se trabaja, la codificación por defecto, si se realiza log de las operaciones en algún fichero, etc.

9. Ver la configuración y el estado del servidor

Desde esta opción podemos ver las sesiones, los bloqueos y las transacciones pendientes. Si además, estamos ejecutando pgAdmin III desde una consola gráfica en el servidor de base de datos, podremos ver el log del servidor, así como, desde una opción del menú Archivo, podemos ver los ficheros de configuración del servidor.

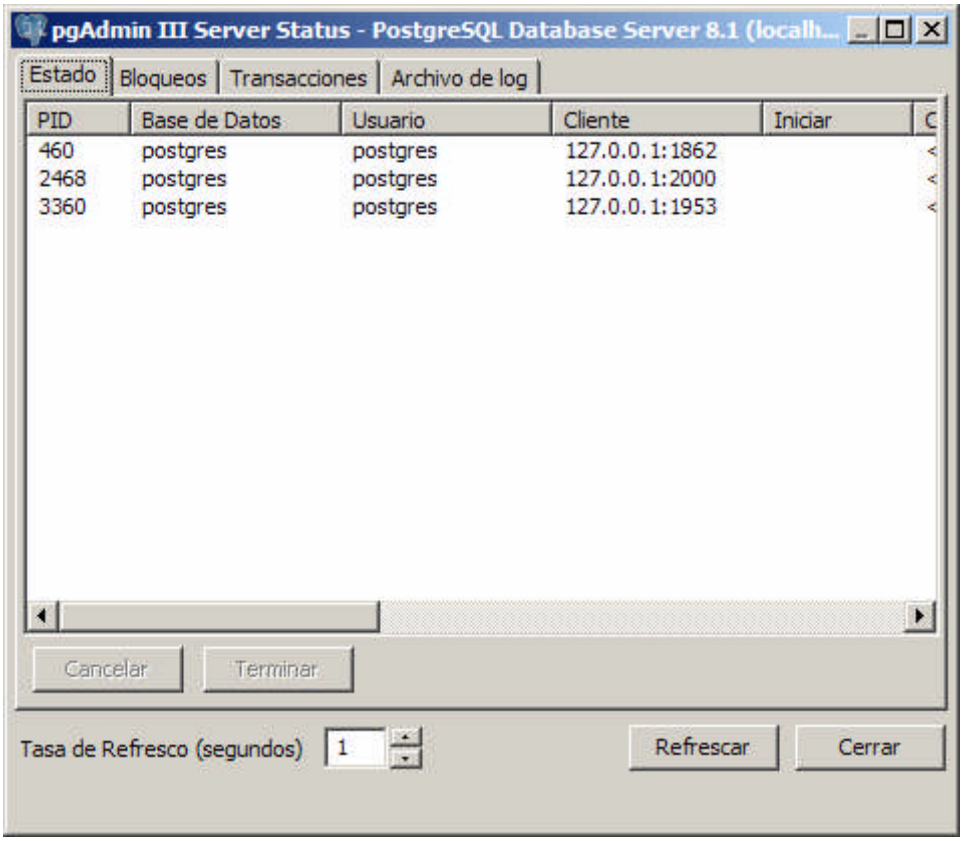

Figura B-12: Estado del servidor

Apéndice C. El Lenguaje SQL de PostgreSQL

Sacado de un tutorial de Postgres en http://es.tldp.org/Postgresql-es/web/navegable/tutorial/sql-language.html

Como en el caso de los más modernos lenguajes relacionales, SQL está basado en el cálculo relacional de tuplas. Como resultado, toda consulta formulada utilizando el cálculo relacional de tuplas (o su equivalente, el álgebra relacional) se pude formular también utilizando SQL. Hay, sin embargo, capacidades que van más allá del cálculo o del álgebra relaciona. Aquí tenemos una lista de algunas características proporcionadas por SQL que no forman parte del álgebra y del cálculo relacionales:

- Comandos para inserción, borrado o modificación de datos.
- Capacidades aritméticas: En SQL es posible incluir operaciones aritméticas así como comparaciones, por ejemplo A < B + 3. Nótese que ni + ni otros operadores aritméticos aparecían en el álgebra relacional ni en cálculo relacional.
- Asignación y comandos de impresión: es posible imprimir una relación construida por una consulta y asignar una relación calculada a un nombre de relación.
- Funciones agregadas: Operaciones tales como *promedio (average)*, *suma (sum)*, *máximo (max)*, etc. se pueden aplicar a las columnas de una relación para obtener una cantidad única.

C.1 Consultas - Select

El comando más usado en SQL es la instrucción SELECT, que se utiliza para recuperar datos. La sintaxis es:

```
SELECT [ALL|DISTINCT]
       { * | expr 1 [AS c alias 1] [, ...
             [, expr k [AS c alias k]]]}
FROM table_name_1 [t_alias_1]
     [, ... [, table name n [t alias n]]]
[WHERE condition]
[GROUP BY name_of_attr_i
          [,... [, name of attr j]] [HAVING condition]]
[{UNION [ALL] | INTERSECT | EXCEPT} SELECT ...]
[ORDER BY name of attr i [ASC|DESC]
          [, ... [, name of attr j [ASC|DESC]]]];
```

Ilustraremos ahora la compleja sintaxis de la instrucción SELECT con varios ejemplos.

Select sencillas

Aquí tenemos algunos ejemplos sencillos utilizando la instrucción SELECT:

Ejemplo 4. Query sencilla con cualificación

Para recuperar todas las tuplas de la tabla PART donde el atributo PRICE es mayor que 10, formularemos la siguiente consulta:

```
SELECT * FROM PART
  WHERE PRICE > 10;
```

y obtenemos la siguiente tabla:

```
 PNO |  PNAME      |  PRICE
-----+-------------+--------
  3  |  Cerrojos   |   15
  4  |  Levas      |   25
```

Utilizando "*" en la instrucción SELECT solicitaremos todos los atributos de la tabla. Si queremos recuperar sólo los atributos PNAME y PRICE de la tabla PART utilizaremos la instrucción:

```
SELECT PNAME, PRICE
FROM PART
WHERE PRICE > 10;
```

En este caso el resultado es:

```
 PNAME      |  PRICE
------------+--------
 Cerrojos   |   15
 Levas      |   25
```

Nótese que la SELECT SQL corresponde a la "proyección" en álgebra relaciona, no a la "selección" (vea *Álgebra Relacional* para más detalles).

Las cualificaciones en la clausula WHERE pueden también conectarse lógicamente utilizando las palabras claves OR, AND, y NOT:

```
SELECT PNAME, PRICE
FROM PART
WHERE PNAME = 'Cerrojos' AND
      (PRICE = 0 OR PRICE < 15);
```

dará como resultado:

```
 PNAME      |  PRICE
------------+--------
 Cerrojos   |   15
```

Las operaciones aritméticas se pueden utilizar en la lista de objetivos y en la clausula WHERE. Por ejemplo, si queremos conocer cuanto cuestan si tomamos dos piezas de un artículo, podríamos utilizar la siguiente consulta:

```
SELECT PNAME, PRICE * 2 AS DOUBLE
FROM PART
WHERE PRICE * 2 < 50;
```

y obtenemos:

```
PNAME      |  DOUBLE
-----------+---------
Tornillos  |    20
Tuercas    |    16
Cerrojos   |    30
```

Nótese que la palabra DOBLE tras la palabra clave AS es el nuevo título de la segunda columna. Esta técnica puede utilizarse para cada elemento de la lista objetivo para asignar un nuevo título a la columna resultante. Este nuevo título recibe el calificativo de "un alias". El alias no puede utilizarse en todo el resto de la consulta.

Joins (Cruces)

El siguiente ejemplo muestra como las *joins (cruces)* se realizan en SQL.

Para cruzar tres tablas SUPPLIER, PART y SELLS a través de sus atributos comunes, formularemos la siguiente instrucción:

```
SELECT S.SNAME, P.PNAME
FROM SUPPLIER S, PART P, SELLS SE
WHERE S.SNO = SE.SNO AND
      P.PNO = SE.PNO;
```

y obtendremos la siguiente tabla como resultado:

```
SNAME | PNAME
-------+-------
Smith | Tornillos
Smith | Tuercas
Jones | Levas
Adams | Tornillos
Adams | Cerrojos
Blake | Tuercas
Blake | Cerrojos
Blake | Levas
```

En la clausula FROM hemos introducido un alias al nombre para cada relación porque hay atributos con nombre común (SNO y PNO) en las relaciones. Ahora podemos distinguir entre los atributos con nombre común simplificando la adicción de un prefijo al nombre del atributo con el nombre del alias seguido de un punto. La join se calcula de la misma forma, tal como se muestra en *Una Inner Join (Una Join Interna)*. Primero el producto cartesiano: SUPPLIER × PART × SELLS Ahora seleccionamos únicamente aquellas tuplas que satisfagan las condiciones dadas en la clausula WHERE (es decir, los atributos con nombre común deben ser iguales). Finalmente eliminamos las columnas repetidas (S.SNAME, P.PNAME).

Operadores Agregados

SQL proporciona operadores agregados (como son AVG, COUNT, SUM, MIN, MAX) que toman el nombre de un atributo como argumento. El valor del operador agregado se calcula sobre todos los valores de la columna especificada en la tabla completa. Si se especifican grupos en la consulta, el cálculo se hace sólo sobre los valores de cada grupo (vean la siguiente sección).

Ejemplo 5. Aggregates

Si queremos conocer el coste promedio de todos los artículos de la tabla PART, utilizaremos la siguiente consulta:

```
SELECT AVG(PRICE) AS AVG_PRICE FROM PART;
```

El resultado es:

```
 AVG_PRICE
-----------
   14.5
```

Si queremos conocer cuantos artículos se recogen en la tabla PART, utilizaremos la instrucción:

```
SELECT COUNT(PNO)
FROM PART;
```

y obtendremos:

```
 COUNT
-------
   4
```

Agregación por Grupos

SQL nos permite particionar las tuplas de una tabla en grupos. En estas condiciones, los operadores agregados descritos antes pueden aplicarse a los grupos (es decir, el valor del operador agregado no se calculan sobre todos los valores de la columna especificada, sino sobre todos los valores de un grupo. El operador agregado se calcula individualmente para cada grupo).

El particionamiento de las tuplas en grupos se hace utilizando las palabras clave **GROUP BY** seguidas de una lista de atributos que definen los grupos. Si tenemos **GROUP BY $A_1, \ldots, A_k$** habremos particionado la relación en grupos, de tal modo que dos tuplas son del mismo grupo si y sólo si tienen el mismo valor en sus atributos $A_1, \ldots, A_k$.

Ejemplo 6. Agregados

Si queremos conocer cuántos artículos han sido vendidos por cada proveedor formularemos la consulta:

```
SELECT S.SNO, S.SNAME, COUNT(SE.PNO)
FROM SUPPLIER S, SELLS SE
WHERE S.SNO = SE.SNO
GROUP BY S.SNO, S.SNAME;
```

y obtendremos:

```
 SNO | SNAME | COUNT
-----+-------+-------
  1  | Smith |   2
  2  | Jones |   1
  3  | Adams |   2
  4  | Blake |   3
```

Demos ahora una mirada a lo que está ocurriendo aquí. Primero, la join de las tablas SUPPLIER y SELLS:

```
 S.SNO | S.SNAME | SE.PNO
-------+---------+--------
   1   |  Smith  |   1
   1   |  Smith  |   2
   2   |  Jones  |   4
   3   |  Adams  |   1
   3   |  Adams  |   3
   4   |  Blake  |   2
   4   |  Blake  |   3
   4   |  Blake  |   4
```

Ahora particionamos las tuplas en grupos reuniendo todas las tuplas que tiene el mismo atributo en S.SNO y S.SNAME:

```
 S.SNO | S.SNAME | SE.PNO
-------+---------+--------
   1   |  Smith  |   1
                 |   2
-------------------------
   2   |  Jones  |   4
-------------------------
   3   |  Adams  |   1
                 |   3
-------------------------
   4   |  Blake  |   2
                 |   3
                 |   4
```

En nuestro ejemplo, obtenemos cuatro grupos y ahora podemos aplicar el operador agregado COUNT para cada grupo, obteniendo el resultado total de la consulta dada anteriormente.

Nótese que para el resultado de una consulta utilizando GROUP BY y operadores agregados para dar sentido a los atributos agrupados, debemos primero obtener la lista objetivo. Los demás atributos que no aparecen en la clausula GROUP BY se seleccionarán utilizando una función agregada. Por otro lado, no se pueden utilizar funciones agregadas en atributos que aparecen en la clausula GROUP BY.

Having

La clausula HAVING trabaja de forma muy parecida a la clausula WHERE, y se utiliza para considerar sólo aquellos grupos que satisfagan la cualificación dada en la misma. Las expresiones permitidas en la clausula HAVING deben involucrar funcionen agregadas. Cada expresión que utilice sólo atributos planos deberá recogerse en la clausula WHERE. Por otro lado, toda expresión que involucre funciones agregadas debe aparecer en la clausula HAVING.

Ejemplo 7. Having

Si queremos solamente los proveedores que venden más de un artículo, utilizaremos la consulta:

```
SELECT S.SNO, S.SNAME, COUNT(SE.PNO)
FROM SUPPLIER S, SELLS SE
WHERE S.SNO = SE.SNO
GROUP BY S.SNO, S.SNAME
HAVING COUNT(SE.PNO) > 1;
```

y obtendremos:

```
 SNO | SNAME | COUNT
-----+-------+-------
  1  | Smith |   2
  3  | Adams |   2
  4  | Blake |   3
```

Subconsultas

En las clausulas WHERE y HAVING se permite el uso de subconsultas (subselects) en cualquier lugar donde se espere un valor. En este caso, el valor debe derivar de la evaluación previa de la subconsulta. El uso de subconsultas amplía el poder expresivo de SQL.

Ejemplo 8. Subselect

Si queremos conocer los artículos que tienen mayor precio que el artículo llamado 'Tornillos', utilizaremos la consulta:

```
SELECT *
FROM PART
WHERE PRICE > (SELECT PRICE FROM PART
               WHERE PNAME='Tornillos');
```

El resultado será:

```
 PNO |  PNAME      |  PRICE
-----+-------------+--------
  3  |  Cerrojos   |   15
  4  |  Levas      |   25
```

Cuando revisamos la consulta anterior, podemos ver la palabra clave SELECT dos veces. La primera al principio de la consulta - a la que nos referiremos como la SELECT externa - y la segunda en la clausula WHERE, donde empieza una consulta anidada - nos referiremos a ella como la SELECT interna. Para cada tupla de la SELECT externa, la SELECT interna deberá ser evaluada. Tras cada evaluación, conoceremos el precio de la tupla llamada 'Tornillos', y podremos chequear si el precio de la tupla actual es mayor.

Si queremos conocer todos los proveedores que no venden ningún artículo (por ejemplo, para poderlos eliminar de la base de datos), utilizaremos:

```
SELECT *
FROM SUPPLIER S
WHERE NOT EXISTS
          (SELECT * FROM SELLS SE
           WHERE SE.SNO = S.SNO);
```

En nuestro ejemplo, obtendremos un resultado vacío, porque cada proveedor vende al menos un artículo. Nótese que utilizamos S.SNO de la SELECT externa en la clausula WHERE de la SELECT interna. Como hemos descrito antes, la subconsulta se evalúa para cada tupla de la consulta externa, es decir, el valor de S.SNO se toma siempre de la tupla actual de la SELECT externa.

Unión, Intersección, Excepción

Estas operaciones calculan la unión, la intersección y la diferencia de la teoría de conjuntos de las tuplas derivadas de dos subconsultas.
Ejemplo 9. Union, Intersect, Except

La siguiente consulta es un ejemplo de UNION:

```
SELECT S.SNO, S.SNAME, S.CITY
FROM SUPPLIER S
WHERE S.SNAME = 'Jones'
UNION
SELECT S.SNO, S.SNAME, S.CITY
FROM SUPPLIER S
WHERE S.SNAME = 'Adams';
```

Dará el resultado:

```
 SNO | SNAME |  CITY
-----+-------+--------
  2  | Jones | Paris
  3  | Adams | Vienna
```

Aquí tenemos un ejemplo para INTERSECT:

```
SELECT S.SNO, S.SNAME, S.CITY
FROM SUPPLIER S
WHERE S.SNO > 1
INTERSECT
SELECT S.SNO, S.SNAME, S.CITY
FROM SUPPLIER S
WHERE S.SNO > 2;
```

que dará como resultado:

```
                    SNO | SNAME |  CITY
                   -----+-------+--------
                     2  | Jones | Paris
        La única tupla devuelta por ambas partes de la consulta es la única que
tiene $SNO=2$.
```

Finalmente, un ejemplo de EXCEPT:

```
   SELECT S.SNO, S.SNAME, S.CITY
   FROM SUPPLIER S
   WHERE S.SNO > 1
   EXCEPT
   SELECT S.SNO, S.SNAME, S.CITY
   FROM SUPPLIER S
   WHERE S.SNO > 3;
```

que dará como resultado:

```
                    SNO | SNAME |  CITY
                   -----+-------+--------
                     2  | Jones | Paris
                     3  | Adams | Vienna
```

C.2 Definición de Datos

El lenguaje SQL incluye un conjunto de comandos para definición de datos.

Create Table

El comando fundamental para definir datos es el que crea una nueva relación (una nueva tabla). La sintaxis del comando CREATE TABLE es:

```
   CREATE TABLE table name
               (name of attr 1 type of attr 1
                [, name_of_attr_2 type_of_attr_2
                [, ...]]);
```

Ejemplo 10. Creación de una tabla

Para crear las tablas definidas en *La Base de Datos de Proveedores y Artículos* se utilizaron las siguientes instrucciones de SQL:

```
   CREATE TABLE SUPPLIER
               (SNO   INTEGER,
                SNAME VARCHAR(20),
                CITY  VARCHAR(20));

   CREATE TABLE PART
```

```
              (PNO   INTEGER,
               PNAME VARCHAR(20),
               PRICE DECIMAL(4 , 2));

CREATE TABLE SELLS
              (SNO INTEGER,
               PNO INTEGER);
```

Tipos de Datos en SQL

A continuación sigue una lista de algunos tipos de datos soportados por SQL:

- INTEGER: entero binario con signo de palabra completa (31 bits de precisión).
- SMALLINT: entero binario con signo de media palabra (15 bits de precisión).
- DECIMAL (p[,q]): número decimal con signo de p dígitos de precisión, asumiendo q a la derecha para el punto decimal. ($15 \geq p \geq q$q ≥ 0). Si q se omite, se asume que vale 0.
- FLOAT: numérico con signo de doble palabra y coma flotante.
- CHAR(n): cadena de caracteres de longitud fija, de longitud n.
- VARCHAR(n): cadena de caracteres de longitud variable, de longitud máxima n.

Create Index

Se utilizan los índices para acelerar el acceso a una relación. Si una relación R tiene un índice en el atributo A podremos recuperar todas la tuplas t que tienen t(A) = a en un tiempo aproximadamente proporcional al número de tales tuplas t más que en un tiempo proporcional al tamaño de R.

Para crear un índice en SQL se utiliza el comando CREATE INDEX. La sintaxis es:

```
CREATE INDEX index name
ON table name ( name of attribute );
```

Ejemplo 11. Create Index

Para crear un índice llamado I sobre el atributo SNAME de la relación SUPPLIER, utilizaremos la siguiente instrucción:

```
CREATE INDEX I
ON SUPPLIER (SNAME);
```

El índice creado se mantiene automáticamente. es decir, cada vez que una nueva tupla se inserte en la relación SUPPLIER, se adaptará el índice I. Nótese que el único cambio que un usuario puede percibir cuando se crea un índice es un incremento en la velocidad.

Create View

Se puede ver una vista como una *tabla virtual*, es decir, una tabla que no existe *físicamente* en la base de datos, pero aparece al usuario como si existiese. Por contra, cuando hablamos de una *tabla base*, hay realmente un equivalente almacenado para cada fila en la tabla en algún sitio del almacenamiento físico.

Las vistas no tienen datos almacenados propios, distinguibles y físicamente almacenados. En su lugar, el sistema almacena la definición de la vista (es decir, las reglas para acceder a las tablas base físicamente almacenadas para materializar la vista) en algún lugar de los catálogos del sistema (vea *System Catalogs*). Para una discusión de las diferentes técnicas para implementar vistas, refiérase a *SIM98*.

En SQL se utiliza el comando **CREATE VIEW** para definir una vista. La sintaxis es:

```
CREATE VIEW view name
AS select stmt
```

donde `select_stmt` es una instrucción select válida, como se definió en *Select*. Nótese que `select_stmt` no se ejecuta cuando se crea la vista. Simplemente se almacena en los *catálogos del sistema* y se ejecuta cada vez que se realiza una consulta contra la vista.

Sea la siguiente definición de una vista (utilizamos de nuevo las tablas de *La Base de Datos de Proveedores y Artículos*):

```
CREATE VIEW London_Suppliers
   AS SELECT S.SNAME, P.PNAME
      FROM SUPPLIER S, PART P, SELLS SE
      WHERE S.SNO = SE.SNO AND
            P.PNO = SE.PNO AND
            S.CITY = 'London';
```

Ahora podemos utilizar esta *relación virtual* `London_Suppliers` como si se tratase de otra tabla base:

```
SELECT *
FROM London_Suppliers
WHERE P.PNAME = 'Tornillos';
```

Lo cual nos devolverá la siguiente tabla:

```
 SNAME | PNAME
-------+----------
 Smith | Tornillos
```

Para calcular este resultado, el sistema de base de datos ha realizado previamente un acceso *oculto* a las tablas de la base SUPPLIER, SELLS y PART. Hace esto ejecutando la consulta dada en la definición de la vista contra aquellas tablas base. Tras eso, las cualificaciones adicionales (dadas en la consulta contra la vista) se podrán aplicar para obtener la tabla resultante.

Drop Table, Drop Index, Drop View

Se utiliza el comando DROP TABLE para eliminar una tabla (incluyendo todas las tuplas almacenadas en ella):

```
DROP TABLE table_name;
```

Para eliminar la tabla SUPPLIER, utilizaremos la instrucción:

```
DROP TABLE SUPPLIER;
```

Se utiliza el comando DROP INDEX para eliminar un índice:

```
DROP INDEX index_name;
```

Finalmente, eliminaremos una vista dada utilizando el comando DROP VIEW:

```
DROP VIEW view_name;
```

C.3 Manipulación de Datos

Insert Into

Una vez que se crea una tabla (vea *Create Table*), puede ser llenada con tuplas mediante el comando **INSERT INTO**. La sintaxis es:

```
INSERT INTO table_name (name_of_attr_1
                        [, name of attr 2 [,...]])
VALUES (val attr 1
        [, val attr 2 [, ...]]);
```

Para insertar la primera tupla en la relación SUPPLIER (de *La Base de Datos de Proveedores y Artículos*) utilizamos la siguiente instrucción:

```
INSERT INTO SUPPLIER (SNO, SNAME, CITY)
VALUES (1, 'Smith', 'London');
```

Para insertar la primera tupla en la relación SELLS, utilizamos:

```
INSERT INTO SELLS (SNO, PNO)
VALUES (1, 1);
```

Update

Para cambiar uno o más valores de atributos de tuplas en una relación, se utiliza el comando UPDATE. La sintaxis es:

```
UPDATE table name
SET name of attr 1 = value 1
    [, ... [, name of attr k = value k]]
WHERE condition;
```

Para cambiar el valor del atributo PRICE en el artículo 'Tornillos' de la relación PART, utilizamos:

```
UPDATE PART
SET PRICE = 15
WHERE PNAME = 'Tornillos';
```

El nuevo valor del atributo PRICE de la tupla cuyo nombre es 'Tornillos' es ahora 15.

Delete

Para borrar una tupla de una tabla particular, utilizamos el comando DELETE FROM. La sintaxis es:

```
DELETE FROM table name
WHERE condition;
```

Para borrar el proveedor llamado 'Smith' de la tabla SUPPLIER, utilizamos la siguiente instrucción:

```
DELETE FROM SUPPLIER
WHERE SNAME = 'Smith';
```

C.4 System Catalogs

En todo sistema de base de datos SQL se emplean *catálogos de sistema* para mantener el control de qué tablas, vistas, índices, etc están definidas en la base de datos. Estos catálogos del sistema se pueden investigar como si de cualquier otra relación normal se tratase. Por ejemplo, hay un catálogo utilizado para la definición de vistas. Este catálogo almacena la consulta de la definición de la vista. Siempre que se hace una consulta contra la vista, el sistema toma primero la *consulta de definición de la vista* del catálogo y materializa la vista antes de proceder con la consulta del usuario (vea *SIM98* para obtener una descripción más detallada). Diríjase a*DATE* para obtener más información sobre los catálogos del sistema.

C.5 SQL Embebido

En esta sección revisaremos como se puede embeber SQL en un lenguaje de host (p.e. `C`). Hay dos razones principales por las que podríamos querer utilizar SQLdesde un lenguaje de host:

- Hay consultas que no se pueden formular utilizando SQL puro (por ejemplo, las consultas recursivas). Para ser capaz de realizar esas consultas necesitamos un lenguaje de host de mayor poder expresivo que SQL.

- Simplemente queremos acceder a una base de datos desde una aplicación que está escrita en el lenguaje del host (p.e. un sistema de reserva de billetes con una interface gráfica escrita en C, y la información sobre los billetes está almacenada en una base de datos que puede accederse utilizando SQL embebido).

Un programa que utiliza SQL embebido en un lenguaje de host consiste en instrucciones del lenguaje del host e instrucciones de *SQL embebido* (ESQL). Cada instrucción de ESQL empieza con las palabras claves **EXEC SQL**. Las instrucciones ESQL se transforman en instrucciones del lenguaje del host mediante un *precompilador* (que habitualmente inserta llamadas a rutinas de librerías que ejecutan los variados comandos de SQL).

Cuando vemos los ejemplos de *Select* observamos que el resultado de las consultas es algo muy próximo a un conjunto de tuplas. La mayoría de los lenguajes de host no están diseñados para operar con conjuntos, de modo que necesitamos un mecanismo para acceder a cada tupla única del conjunto de tuplas devueltas por una instrucción SELECT. Este mecanismo puede ser proporcionado declarando un *cursor*. Tras ello, podemos utilizar el comando FETCH para recuperar una tupla y apuntar el cursor hacia la siguiente tupla.

C.6 Características Avanzadas de SQL en Postgres

Habiendo cubierto los aspectos básicos de Postgre SQLpara acceder a los datos, discutiremos ahora aquellas características de Postgres que los distinguen de los gestores de bases de datos convecionales. Estas características incluyen herencia, time travel (viaje en el tiempo) y valores no-atómicos de datos (atributos basados en vectores y conjuntos). Los ejemplos de esta sección pueden encontrarse también en `advance.sql` en el directorio del tutorial. (Consulte el el capítulo de nombre *El Lenguaje de consultas* para ver la forma de utilizarlo).

Herencia

Creemos dos clases. La clase capitals contiene las capitales de los estados, las cuales son también ciudades. Naturalmente, la clase capitals debería heredar de cities.

```
CREATE TABLE cities (
    name            text,
    population      float,
    altitude        int     -- (in ft)
);

CREATE TABLE capitals (
    state           char(2)
) INHERITS (cities);
```

En este caso, una instancia de capitals *hereda* todos los atributos (name, population y altitude) de su padre, cities. El tipo del atributo name (nombre) es text, un tipo nativo de Postgres para cadenas ASCII de longitud variable. El tipo del atributo population (población) es float, un tipo de datos, también nativo de Postgres , para números de punto flotante de doble precisión. Las clase capitals tiene un atributo extra, state, que muestra a qué estado pertenecen. En Postgres, una clase puede heredar de ninguna o varias otras clases, y una consulta puede hacer referencia tanto a todas las instancias de una clase como a todas las instancias de una clase y sus descendientes.
Por ejemplo, la siguiente consulta encuentra todas aquellas ciudades que están situadas a un altura de 500 o más pies:

```
SELECT name, altitude
    FROM cities
    WHERE altitude > 500;

+----------+----------+
|name      | altitude |
+----------+----------+
|Las Vegas | 2174     |
+----------+----------+
|Mariposa  | 1953     |
+----------+----------+
```

Por otro lado, para encontrar los nombres de todas las ciudades, incluídas las capitales estatales, que estén situadas a una altitud de 500 o más pies, la consulta es:

```
SELECT c.name, c.altitude
    FROM cities* c
    WHERE c.altitude > 500;
```

que devuelve:

```
|name      | altitude |
+----------+----------+
|Las Vegas | 2174     |
+----------+----------+
|Mariposa  | 1953     |
+----------+----------+
|Madison   | 845      |
+----------+----------+
```

Aquí el "*" después de cities indica que la consulta debe realizarse sobre cities y todas las clases que estén por debajo de ella en la jerarquía de la herencia. Muchos de los comandos que ya hemos discutido (**select**, **and>up**and> and **delete**) brindan soporte a esta notación de "*" al igual que otros como **alter**.

Valores No-Atómicos

Uno de los principios del modelo relacional es que los atributos de una relación son atómicos Postgres no posee esta restricción; los atributos pueden contener sub-valores a los que puede accederse desde el lenguaje de consulta. Por ejemplo, se pueden crear atributos que sean vectores de alguno de los tipos base.

Vectores

Postgres permite que los atributos de una instancia sean definidos como vectores multidimensionales de longitud fija o variable. Puede crear vectores de cualquiera de los tipos base o de tipos definidos por el usuario. Para ilustrar su uso, creemos primero una clase con vectores de tipos base.

```
CREATE TABLE SAL_EMP (
    name            text,
    pay_by_quarter  int4[],
    schedule        text[][]
);
```

La consulta de arriba creará una clase llamada SAL_EMP con una cadena del tipo *text* (name),un vector unidimensional del tipo *int4* (pay_by_quarter), el cual representa el salario trimestral del empleado y un vector bidimensional del tipo *text* (schedule), que representa la agenda semanal del empleado. Ahora realizamos algunos *INSERTS*s; note que cuando agregamos valores a un vector, encerramos los valores entre llaves y los separamos mediante comas. Si usted conoce *C*, esto no es distinto a la sintaxis para inicializar estructuras.

```
INSERT INTO SAL_EMP
    VALUES ('Bill',
    '{10000, 10000, 10000, 10000}',
    '{{"meeting", "lunch"}, {}}');

INSERT INTO SAL_EMP
    VALUES ('Carol',
    '{20000, 25000, 25000, 25000}',
    '{{"talk", "consult"}, {"meeting"}}');
```

Postgres utiliza de forma predeterminada la convención de vectores "basados en uno" -- es decir, un vector de n elementos comienza con vector[1] y termina con vector[n]. Ahora podemos ejecutar algunas consultas sobre SAL_EMP. Primero mostramos como acceder a un solo elemento del vector por vez. Esta consulta devuelve los nombres de los empleados cuyos pagos han cambiado en el segundo trimestre:

```
SELECT name FROM SAL_EMP
    WHERE SAL_EMP.pay_by_quarter[1] <> SAL_EMP.pay_by_quarter[2];
+------+
|name  |
+------+
|Carol |
```

La siguiente consulta recupera el pago del tercer trimestre de todos los empleados:

```
SELECT SAL_EMP.pay_by_quarter[3] FROM SAL_EMP;


+---------------+
|pay_by_quarter |
+---------------+
|10000          |
+---------------+
|25000          |
+---------------+
```

También podemos acceder a cualquier porción de un vector, o subvectores. Esta consulta recupera el primer item de la agenda de Bill para los primeros dos días de la semana.

```
SELECT SAL_EMP.schedule[1:2][1:1]
    FROM SAL_EMP
    WHERE SAL_EMP.name = 'Bill';

+-------------------+
|schedule           |
+-------------------+
```

```
|{{"meeting"},{""}} |
+-------------------+
```

Bibliografia y referencias

Libros, apuntes y cursos

Estos apuntes se han basado principalmente en los siguientes materiales:

PostgreSQL 8.2.5 Documentación [http://www.postgresql.org]

Curso de Administración de PostgreSQL
Juan Carlos Casamayor
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Curso de Administración de PostgreSQL
Alvaro Herrera (Publicado en Internet)

Curso de PostgreSQL, versión 6.5
Equipo de Desarrollo de PostgreSQL (Publicado en Internet)

Manual de Usuario de PostgreSQL versión 6.5
Equipo de Desarrollo de PostgreSQL Editado por Thomas Lockhart

Mastering PostgreSQL Administration
Bruce Momjian
Software Research Associates

TLDP-LUCAS [http://es.tldp.org/Postgresql-es/web/]

Enlaces Internet

Sitios oficiales de PostgreSQL

Sitio Oficial [http://www.postgresql.org]

Comunidad de soporte

- Listas oficiales de correo [http://www.postgresql.org/community/lists/]
- Suscripción a las listas oficiales [http://mail.postgresql.org/mj/mj_wwwusr?domain=postgresql.org&func=lists-long-full&extra=pgsql-es-ayuda]

Proyectos PostgreSQL

- GBorg (psqlODBC) [http://gborg.postgresql.org/project/psqlodbc/faq/faq.php]
- pgFoundry [http://pgfoundry.org]
- GiST y tsearch2:
    - Megera [http://www.sai.msu.su/~megera/postgres/gist/]
    - Berkeley [http://gist.cs.berkeley.edu/pggist/]

- PostGis [http://www.postgis.org/]
- slony [http://slony.info/]
- pgAdmin III [http://www.pgadmin.org]
- phppgAdmin III [http://phppgadmin.sourceforge.net]
- pgExplorer [http://www.pgexplorer.org]
- pgAccess [http://www.pgaccess.org]
- pgCluster [http://users.servicios.retecal.es/tjavier/presbbdd/out-htmls/x177.html]

Empresas

- Open Minds (Cluster BD y Alta Disponibilidad)
  [http://www.openminds.co.uk/high_availability_solutions/databases/postgresql.htm]